

# Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

## Diplomarbeit

### Netzwerkinventarisierung und -dokumentation Design und Implementierung einer plattformunabhängigen Softwarelösung

Leipzig, August 1998

Thomas Fleischer

Betreut durch:      Herrn Prof. Dr.-Ing. W. G. Spruth  
                             Herrn M. Frese

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis .....</b>	<b>2</b>
<b>Abbildungsverzeichnis .....</b>	<b>5</b>
<b>Tabellenverzeichnis .....</b>	<b>6</b>
<b>1 Einleitung .....</b>	<b>7</b>
1.1 Problemstellung und Zielsetzung .....	8
1.2 Gliederung der Arbeit .....	9
<b>2 Grundlagen des Netzwerkmanagements .....</b>	<b>10</b>
2.1 Ausgangsproblem .....	10
2.2 Anforderungen und Ziele .....	11
2.3 Kategorien und Aufgabenbereiche des Netzwerkmanagements .....	12
2.3.1 Fault Management .....	14
2.3.2 Accounting Management .....	15
2.3.3 Configuration Management .....	16
2.3.4 Performance Management .....	17
2.3.5 Security Management .....	18
2.4 Allgemeines Modell des Netzwerkmanagements .....	18
2.4.1 Agenten .....	20
2.4.2 Managed Objects .....	20
2.4.3 Managementarchitektur .....	21
2.4.4 Managementinformationen .....	22
2.4.5 Polling vs. Traps .....	23
<b>3 SNMP - Simple Network Management Protocol .....</b>	<b>25</b>
3.1 Ursprünge des TCP/IP-Netzwerkmanagements .....	25
3.2 Zielsetzung .....	28
3.3 Grundkonzepte von SNMP .....	28
3.3.1 Das Netzmodell .....	28
3.3.2 Die Protokollarchitektur .....	31
3.3.3 Der Proxy-Agent .....	32
3.3.4 Trap-Directed-Polling .....	33
3.4 SNMP-Managementinformationen .....	34
3.4.1 ASN.1 .....	34
3.4.2 ASN.1-Transfersyntax .....	38
3.4.3 SMI .....	40
3.4.4 MIB .....	42
3.5 Protokoll und Operationen .....	45
3.5.1 GetRequest .....	47

3.5.2	GetResponse .....	48
3.5.3	GetNextRequest .....	48
3.5.4	SetRequest .....	49
3.5.5	Trap .....	49
3.6	SNMP Security .....	50
3.6.1	Authentifizierung .....	51
3.6.2	Zugriffsberechtigung .....	51
3.7	Weiterentwicklungen des SNMPv1-Standards .....	52
3.7.1	SNMPv2p .....	53
3.7.2	SNMPv2c .....	56
3.7.3	SNMPv2u .....	56
3.7.4	SNMPv3 .....	56
<b>4</b>	<b>Java .....</b>	<b>57</b>
4.1	Die Entwicklung von Java .....	57
4.2	Exkurs: World Wide Web .....	58
4.3	Das Java-System .....	59
4.3.1	Die Bestandteile .....	59
4.3.2	Die Eigenschaften .....	60
4.3.3	Compiler, Interpreter und Virtuelle Maschine .....	63
4.3.4	Just-In-Time-Compiler (JIT) .....	65
4.3.5	Applikationen und Applets .....	67
4.4	Das Sicherheitssystem von Java .....	69
4.4.1	Das Sprachdesign .....	70
4.4.2	Der Bytecode-Verifier .....	70
4.4.3	Der Klassenlader .....	71
4.4.4	Der Security-Manager .....	72
4.4.5	Die Kosten der Sicherheit .....	72
<b>5</b>	<b>Die Anwendung „JReport“ .....</b>	<b>73</b>
5.1	Das Konzept der Anwendung .....	73
5.1.1	Vorbemerkungen .....	73
5.1.2	Die Entwicklungsumgebung .....	73
5.1.3	Die Struktur der Anwendung .....	74
5.1.4	Der Applikations-Client .....	75
5.1.4.1	Die Benutzeroberfläche .....	76
5.1.4.2	Die Kommunikationsschnittstelle .....	77
5.1.5	Der Applikations-Server .....	80
5.1.5.1	Die Kommunikationsschnittstelle .....	81
5.1.5.2	Der Reportgenerator .....	82

5.1.5.3	Der HTML-Generator .....	84
5.1.6	Die SNMP-Library .....	84
5.1.7	Die MIB-Library .....	87
5.2	Die implementierte Softwarelösung .....	90
5.2.1	Systemanforderungen.....	90
5.2.2	Patches und Workarounds .....	90
5.2.3	Applikationsbeschreibung .....	92
5.2.4	Getestete Umgebungen .....	98
<b>6</b>	<b>Zusammenfassung und Ausblick.....</b>	<b>99</b>
	<b>Abkürzungsverzeichnis .....</b>	<b>102</b>
	<b>Literatur- und Quellenverzeichnis .....</b>	<b>105</b>
	<b>Anhang A: Verwendete Hard- und Software.....</b>	<b>107</b>
	<b>Anhang B: Quelltexte .....</b>	<b>108</b>
	<b>Danksagung .....</b>	<b>109</b>
	<b>Erklärung .....</b>	<b>110</b>

## Abbildungsverzeichnis

Abbildung 1: Austausch von Managementinformationen (nach [JAN93]) .....	19
Abbildung 2: Managementstation und Agent (nach [JAN93]) .....	20
Abbildung 3: Generelle Managementarchitektur (nach [JAN93]) .....	22
Abbildung 4: SNMP-Netzmodell (nach [STA96]) .....	29
Abbildung 5: Protokollkontext von SNMP (nach [LIN96]) .....	31
Abbildung 6: Proxy-Konfiguration (nach [SLO94]) .....	33
Abbildung 7: Hierarchiebaum für ASN.1-Objekte .....	36
Abbildung 8: Bitordnung nach BER .....	38
Abbildung 9: Tag-Feld nach BER .....	39
Abbildung 10: Beispiele für BER-Kodierungen .....	40
Abbildung 11: Beispiel einer Objektdeklaration .....	42
Abbildung 12: Ablauf der SNMP-Operationen .....	46
Abbildung 13: SNMP-Nachrichtenformate (nach [STA96]) .....	46
Abbildung 14: Vergleich von Java mit compilierten und interpretierten Sprachen .....	65
Abbildung 15: Aufbau der unterschiedlichen JVMs .....	67
Abbildung 16: Beispiel einer Java-Applikation .....	68
Abbildung 17: Beispiel eines Java-Applets .....	68
Abbildung 18: Struktur der Anwendung JReport .....	74
Abbildung 19: Aufbau des Applikations-Clients .....	75
Abbildung 20: Aufbau einer RPC-Nachricht .....	79
Abbildung 21: Aufbau des Applikations-Servers .....	80
Abbildung 22: Struktur der SNMP-Library .....	85
Abbildung 23: Struktur der MIB-Library .....	88
Abbildung 24: Die Anwendung JReport - Startbildschirm .....	93
Abbildung 25: Die Anwendung JReport - Auswahl der MIB-Dateien .....	94
Abbildung 26: Die Anwendung JReport - Auswahl der MIB-Objekte .....	95
Abbildung 27: Die Anwendung JReport - Konfiguration der Geräteadressen .....	96
Abbildung 28: Die Anwendung JReport - generierte Berichtsübersicht .....	97
Abbildung 29: Die Anwendung JReport - Beispiel eines erstellten Berichts .....	97

## Tabellenverzeichnis

Tabelle 1: Management Function Standards (nach [SLO94]) .....	13
Tabelle 2: Einfache ASN.1-Datentypen .....	35
Tabelle 3: Verfeinerte Datentypen in der SMI .....	41
Tabelle 4: Objektgruppen der MIB-II .....	45
Tabelle 5: Feldbeschreibungen der SNMP-Nachrichten .....	47
Tabelle 6: Ergänzende Feldbeschreibung der Trap-PDU .....	50
Tabelle 7: Beziehung zwischen Access Mode und ACCESS-Attribut (nach [ROS93]) .....	52
Tabelle 8: Zusätzliche SNMPv2-Datentypen .....	54
Tabelle 9: Layout-Manager in Java .....	77
Tabelle 10: Stubfunktionen des RPC .....	78
Tabelle 11: Klassen der SNMP-Library .....	86
Tabelle 12: Klassen der MIB-Library .....	88
Tabelle 13: Getestete Umgebungen der Applikation JReport .....	98

# 1 Einleitung

In der heutigen Informationsgesellschaft tragen von Computern verwaltete Daten maßgeblich zum Erfolg von Unternehmen bei. Entscheidend für sie ist der schnelle und flexible Zugriff auf nahezu beliebige Daten- und Informationsmengen. Dies erfordert eine Kopplung bestehender und neuer Computersysteme und somit den Einsatz von Netzwerken. Dabei geht der Trend zu immer größeren Netzen, die einer wachsenden Zahl von Anwendern den Zugriff auf Applikationen und Daten ermöglichen. Im Zuge dieser Entwicklung wird ein Fakt offensichtlich: Die Netzwerke werden zu einer unentbehrlichen Ressource für das Unternehmen, deren permanente Verfügbarkeit jedoch durch die steigende Zahl potentieller Fehlerquellen gefährdet ist. Die Komplexität solcher Systeme erfordert daher den Einsatz eines umfassenden Netzwerkmanagements, das die Überwachung und Steuerung aller Netzkomponenten unterstützt. Die Konzepte eines solchen Netzwerkmanagements bilden die Grundlage dieser Arbeit und werden am Beispiel einer Anwendung für die Netzwerkinventarisierung und -dokumentation näher untersucht.

Den Anstoß für das Thema dieser Arbeit gab ein Praktikum bei der IBM Deutschland Informationssysteme GmbH, das in der Kundenumgebung der Norddeutschen Landesbank Hannover (NORD/LB) absolviert wurde. Dort wurde im Rahmen einer Projektarbeit eine Migration des bestehenden Backbone-Netzes in ein umfangreiches ATM-Netzwerk durchgeführt.

Während der schrittweisen Umstellung wurden an der vorhandenen Netzwerktechnik zahlreiche Konfigurationsarbeiten vorgenommen. Davon waren insbesondere die neuen ATM-Komponenten (ATM-Switches, ATM-LAN-Bridges) betroffen, die im Laufe des Projekts z.T. mehrfachen Änderungen unterlagen. Um in der ausgedehnten Netzstruktur den Überblick über die bereits durchgeführten Arbeiten zu bewahren, mußte der jeweils aktuelle Konfigurationsstand aller Netzkomponenten sorgfältig dokumentiert werden. Dies geschah unter Verwendung unterschiedlicher Softwareprodukte, zu denen u.a. das Netzwerkmanagementsystem „Tivoli NetView for AIX R3 V1“, auch kurz „NetView / 6000“ genannt, gehörte. Doch trotz der hochspezialisierten Managementfunktionen dieser Produkte zeigt es sich, daß sie für die eigentliche Dokumentation ungeeignet waren. Die benötigten Informationen mußten für jedes Gerät umständlich einzeln erfaßt werden, da es nicht möglich war, Sammelanfragen über bestimmte Gerätekategorien bzw. -gruppen zu erstellen. Daher wurde von Seiten der NORD/LB angeregt, dieses Problem mit einer kleinen, aber speziellen Software zu lösen. Die Idee stieß auf reges Interesse bei der zuständigen Abteilung der IBM und bildete die Grundlage dieser Arbeit.

## 1.1 Problemstellung und Zielsetzung

Ziel der vorliegenden Arbeit ist die Konzeption und Umsetzung eines Softwarewerkzeuges zur Dokumentation von Netzwerkkomponenten.

Mit der Anwendung „JReport“ soll ein Reportgenerator vorgestellt werden, der es dem Netzwerkadministrator ermöglicht, Informationen auf Basis des Netzwerkmanagementprotokolls SNMPv1 (*Simple Network Management Protocol Version 1*) über bestimmte, konfigurierbare Gerätekategorien (z.B. Bridges, Router und Switches) abzufragen. Die gesammelten Daten sollen aufbereitet und in tabellarischen Berichten chronologisch abgelegt werden. Die Verwendung des Dokumentenformates HTML (*HyperText Markup Language*) soll die Darstellung der Reports mit einem beliebigen Web-Browser ermöglichen, so daß auch Anwender mit unterschiedlicher Hardwareplattform auf diese Informationen zugreifen können.

Die Besonderheit dieser Lösung liegt darin, daß die Applikation selbst von der zugrunde liegenden Architektur unabhängig sein soll. Durch die Verwendung der Programmiersprache und Laufzeitumgebung Java wäre der Einsatz von „JReport“ prinzipiell unter jedem Betriebssystem möglich, für das eine solche Umgebung zur Verfügung steht. Dies würde sich vor allem in Netzwerken mit heterogenen Strukturen anbieten, in denen z.B. unterschiedliche Betriebssystemplattformen (UNIX, MS Windows, OS/2 ...) vorhanden sind. Aber auch die mobile Anwendung im Servicebereich wäre denkbar. Aufgrund der geringen Anforderungen an die Arbeitsumgebung könnte „JReport“ bei Wartungstätigkeiten vor Ort von Servicetechnikern eingesetzt werden. Insgesamt eignet sich eine solche Applikation zur flexiblen Erfassung und Dokumentation von Konfigurationsparametern und Versionsständen aktiver Netzwerkkomponenten.

Diese Arbeit beschreibt die Grundlagen des allgemeinen Netzwerkmanagements und gibt einen detaillierten Einblick in die Konzepte und Funktionen des Managementprotokolls SNMP. Weiterhin setzt sich die Arbeit mit Java auseinander. Am Beispiel der Applikation „JReport“ wird gezeigt, daß mit der gleichnamigen Programmiersprache umfangreiche Aufgabenstellungen bearbeitet werden können und die zugehörige Laufzeitumgebung trotzdem plattform-unabhängige Softwarelösungen ermöglicht. Aufgrund dieser Eigenschaft von Java sind auch Fragen zur Integrität und Sicherheit des lokalen Rechners zu beantworten.

Der Schwerpunkt der Arbeit liegt in der Entwicklung eines Konzeptes für ein System mit obengenanntem Funktionsumfang und der praktischen Umsetzung der gewonnenen Erkenntnisse, d.h. der Implementierung einer portablen Anwendung in Java.



## 1.2 Gliederung der Arbeit

Die Arbeit ist in sechs Kapitel unterteilt, die sich mit folgenden Themen beschäftigen:

Das *Kapitel 1: Einleitung* beschreibt das Thema, die Problemstellung und Zielsetzung sowie die Gliederung der Arbeit.

Das *Kapitel 2: Grundlagen des Netzwerkmanagements* befaßt sich mit der Grundproblematik bei der Verwaltung von Datennetzen und macht mit einigen Anforderungen und Zielen an ein Netzwerkmanagementsystem vertraut. Der Abschnitt enthält weiterhin eine Beschreibung der unterschiedlichen Aufgabenbereiche eines solchen Systems und gibt mit einem allgemeinen Modell des Netzwerkmanagements einen Überblick über verwendete Konzepte und Begriffe.

Im *Kapitel 3: SNMP* wird das gleichnamige Managementprotokoll im Detail beschrieben. Dabei werden zunächst die Grundkonzepte von SNMP vorgestellt und anschließend die Managementinformationen und Protokolloperationen erläutert. Der Abschnitt schließt mit einer Beschreibung des Sicherheitsmodells und einem Überblick über die Weiterentwicklung des SNMP-Standards.

Das *Kapitel 4: Java* setzt sich mit dem Konzept der gleichnamigen Programmiersprache und Laufzeitumgebung auseinander. Dabei werden die Eigenschaften des Java-Systems erläutert und die einzelnen Bestandteile aufgelistet. Der Abschnitt schließt mit einer Beschreibung des integrierten Sicherheitssystems.

Im *Kapitel 5: Die Anwendung „JReport“* werden die Entwicklung eines Konzepts und die Implementierung eines Reportgenerators beschrieben, der die Dokumentation von Parametern und Einstellungen von Netzkomponenten auf der Grundlage des Managementprotokolls SNMPv1 ermöglicht. Dabei werden die einzelnen Bestandteile der Anwendung erläutert und Schwierigkeiten bei der Umsetzung der plattformunabhängigen Lösung aufgezeigt.

Das *Kapitel 6: Zusammenfassung und Ausblick* weist auf mögliche Verbesserungen und Erweiterungen hin.

Der *Anhang A* listet sämtliche Softwarewerkzeuge und die verwendete Hardware auf, die bei der Entwicklung von „JReport“ zum Einsatz kamen.

Der *Anhang B* enthält Angaben zu den Quelltexten der Applikation „JReport“.

In der *Anlage* zu dieser Arbeit befindet sich eine Diskette, die eine ablauffähige Version der Anwendung enthält. Zusätzlich sind sämtliche Quelltexte und eine Kurzbeschreibung mit Hinweisen für eine Installation beigelegt.

## 2 Grundlagen des Netzwerkmanagements

Dieses Kapitel soll einen Einblick in die Problematik des Netzwerkmanagements bzw. der Netzwerkverwaltung geben und so einige Grundkenntnisse für das Verständnis der vorliegenden Arbeit liefern. Eine detaillierte Einführung kann der zahlreichen Fachliteratur zu diesem Thema entnommen werden, z.B. [ROS94], [STA96].

### 2.1 Ausgangsproblem

In vielen Unternehmen sind die Arbeitsplatzrechner heute an ein lokales Netzwerk (*LAN - Local Area Network*) angeschlossen und dort mit verschiedenen Datei- und Datenbankservern oder Netzwerkdruckern verbunden. Dabei wachsen die zugrunde liegenden Netzwerkstrukturen, um einer steigenden Anzahl von Anwendern per Netz den Zugriff auf Applikationen und Daten zu ermöglichen. Die Komplexität solcher Netzwerke nimmt zu, und in manchen Konzernen verwischen mitunter die Unterschiede zwischen herkömmlichen LANs und inzwischen aufgebauten MANs (*Metropolitan Area Network*).

Die neugewonnenen Möglichkeiten bringen jedoch nicht nur Vorteile. Es kommt zu einer zunehmenden Abhängigkeit des Unternehmens vom Netzwerk und den damit verbundenen Ressourcen. Der Ausfall des Netzes oder bestimmter Netzkomponenten hat Einfluß auf weite Teile der internen Kommunikation und kann, durch Störung von Produktionsabläufen, umfangreiche finanzielle Schäden verursachen. Der Zustand des Netzwerkes wird daher für manches Unternehmen „überlebenswichtig“. Unter solchen Bedingungen wird eine permanent hohe Verfügbarkeit, möglichst nahe 100 Prozent, erwartet. Dies läßt sich jedoch durch die alleinige Tätigkeit eines Netzwerkadministrators kaum sicherstellen. Die Komplexität solcher Netzwerke erfordert den Einsatz automatisierter Managementwerkzeuge, die das verantwortliche Personal bei der Überwachung und Steuerung der Systeme unterstützen.

Hinzu kommt noch ein weiteres Problem: Die Heterogenität. In den ehemals homogenen Netzwerken, deren Technik ein einziger Hersteller bereitstellte, werden nach einem Ausbau häufig spezialisierte Geräte unterschiedlicher Anbieter eingesetzt. Dies verschärft die Problematik der Netzwerkverwaltung, da proprietäre Managementlösungen nicht mehr vernünftig angewendet werden können. Der Aufwand, die einzelnen Softwaretools der verschiedenen Hersteller auf dem neuesten Stand zu halten und die Mitarbeiter dafür fortlaufend zu schulen, wäre einfach zu hoch. Dabei ist der Punkt, ob der entsprechende Anbieter eine aktuelle Verwaltungssoftware seiner Geräte, für das vor Ort eingesetzte Betriebssystem, überhaupt liefern kann, noch nicht einmal betrachtet worden.

Um auch in heterogenen Umgebungen die Kontrolle und Steuerung der Netzkomponenten effizient und kostengünstig vornehmen zu können, ist der Einsatz standardisierter Werkzeuge erforderlich. Ein solches Netzwerkmanagementsystem muß daher ein breites Spektrum unter-

schiedlicher Produkte abdecken und so die herstellerunabhängige Verwaltung der Netzressourcen ermöglichen.

Unter Ressourcen werden dabei alle Komponenten eines Netzwerkes verstanden, die zu dessen Funktion beitragen, unabhängig davon, ob sie physischer oder logischer Natur sind. Beispiele für physische Ressourcen sind Geräte, wie Endsysteme (*Hosts*), Router, Bridges, Switches usw., logische Ressourcen sind z.B. Verbindungen und Dienstschnittstellen (*SAP - Service Access Points*).

Neben diesen beiden „klassischen“ Ressourcen gewinnt eine dritte Gruppe zunehmend an Bedeutung: Softwarekomponenten in Form von elektronischen Mail-Systemen, Server-Systemen oder Verzeichnisdiensten (*Directory Services*). Durch den verstärkten Einsatz verteilter Anwendungen wird man sich zukünftig häufiger mit dieser Variante des Netzwerkmanagements auseinandersetzen müssen.

## **2.2 Anforderungen und Ziele**

An ein standardisiertes Netzwerkmanagementsystem (kurz: *NMS*) werden unterschiedliche Anforderungen gestellt. Grundsätzlich ist eine zentralisierte Kontrolle und Steuerung aller vorhandenen Komponenten im Netzwerk erwünscht. Dabei sollen die Geräte fortlaufend überwacht (*Monitoring*) und Zustandsabweichungen per Alarm gemeldet werden. Eine chronologische Aufzeichnung solcher Ereignisse (*Event Logging*) ist für das Aufspüren von Fehlerquellen notwendig. Eine weitere Forderung an das Netzwerkmanagement ist die Absicherung einer permanent hohen Verfügbarkeit aller Netzwerkressourcen. Angestrebt wird eine Verfügbarkeitsquote von knapp 100 Prozent, bezogen auf ein Jahr Laufzeit. Dabei bedeutet eine durchgängige Verfügbarkeit von 99 Prozent immer noch eine Ausfallzeit von ca. 3,6 Tagen bzw. 87 Stunden im Jahr. Das Zahlenbeispiel demonstriert, daß in vielen Unternehmen, insbesondere im Bank- und Finanzwesen, eine weitere Verbesserung dieser Quote von großer Bedeutung ist. Es muß jedoch auch gesagt werden, daß ein NMS kein vernünftiges redundantes Netzwerkdesign ersetzen kann. In Verbindung mit entsprechenden Backupgeräten und alternativen Kommunikationsleitungen trägt es jedoch einen entscheidenden Teil zur Reduktion von Ausfallzeiten bei.

In einigen Unternehmen ist zusätzlich der Aspekt der Kostenkontrolle von Bedeutung. Dabei können die von den Anwendern in Anspruch genommenen Ressourcen für Abrechnungszwecke aufgezeichnet werden. Diese Form der Ressourcenüberwachung, auch *Ressource Monitoring* genannt, gestattet weitere Anwendungen. Mit ihr sind u.a. Lastmessungen in den vorhandenen Netzkomponenten möglich. So lassen sich Engpässe frühzeitig erkennen und die Anforderungen der Anwender durch angemessene Strukturierungsmaßnahmen erfüllen.

Insgesamt wird durch den Einsatz eines Netzwerkmanagementsystems ein verbesserter Service erwartet, so daß auch bei wachsender Netzwerkstruktur die Ressourcen in der gleichen

oder einer höheren Qualität zur Verfügung gestellt werden und ein störungsfreier Zugriff auf Informationen möglich ist.

Aus der Sicht der Netzbetreiber beschreibt Janssen in [JAN93] die Anforderungen an ein standardisiertes Netzwerkmanagement. Ausgangspunkt ist ein heterogenes Umfeld, das sich aus Netzkomponenten unterschiedlicher Hersteller zusammensetzt und für deren Verwaltung proprietäre Managementprotokolle existieren. Trotzdem benötigt der Netzbetreiber eine einheitliche logische Gesamtansicht über das Netz mit allen seinen Komponenten. Das heißt:

- einheitliches Netzmodell,
- einheitliche Präsentation der Netzstruktur,
- einheitliche Grundfunktionen wie Alarmierung und Anzeige von Parametern und Zuständen,
- einheitliche Datenbasis für Komponenten, Zustände und Statistikdaten.

Um diese Anforderungen zu erfüllen, müßte ein standardisiertes NMS die Netzkomponenten direkt verwalten oder als integrierende Schicht über den herstellerspezifischen Managementsystemen liegen.

Auch Janssen kommt zu dem Schluß, daß die Ziele der Netzwerkverwaltung darin bestehen,

- eine konstant hohe Dienstgüte zu gewährleisten,
- eine flexible Anpassung des Netzes an veränderte Anforderungen zu ermöglichen und
- eine effiziente Nutzung der vorhandenen Netzressourcen für ein optimales Kosten-Nutzen-Verhältnis zu erreichen.

## **2.3 Kategorien und Aufgabenbereiche des Netzwerkmanagements**

Eines der standardisierten Konzepte zur Netzwerkverwaltung ist das OSI-Network-Management (*OSI - Open Systems Interconnection*) ([JAN93], [SLO94]). Es wird seit 1985 von der ISO (*International Organization for Standardization*) entwickelt und wurde mit dem OSI Management Framework (ISO 7498-4) erstmals 1989 normiert. Innerhalb dieses Standardisierungsprozesses hat man sich u.a. mit den Aufgaben auseinandergesetzt, die ein solches Managementsystem zu erbringen hat. Dazu wurde das Gebiet in fünf Funktionsbereiche (*SMFAs - Systems Management Functional Areas*) aufgeteilt.

Das sind:

- Fault Management (Fehlermanagement),
- Accounting Management (Rechnungsmanagement),
- Configuration Management (Konfigurationsmanagement),
- Performance Management (Leistungsmanagement) und
- Security Management (Sicherheitsmanagement).

Die SMFAs beschreiben die allgemeinen Anforderungen, die zur Bewältigung der Aufgaben der einzelnen Funktionsbereiche erfüllt werden müssen. Innerhalb jeder SMFA definieren die Normen ISO 10164-x die eigentlichen Details (siehe dazu Tabelle 1).

<b>ISO Referenz</b>	<b>CCITT Referenz</b>	<b>Titel</b>	<b>SMFA</b>
IS 10164-1	X.730	Object Management Function	C
IS 10164-2	X.731	State Management Function	C
IS 10164-3	X.732	Objects and Attributs for Representing Relationships	C
IS 10164-4	X.733	Alarm Reporting Function	F
IS 10164-5	X.734	Event Report Management Function	F
IS 10164-6	X.735	Log Control Function	F
IS 10164-7	X.736	Security Alarm Reporting Function	S
IS 10164-8	X.740	Security Audit Trail Function	S
DIS 10164-9	X.741	Objects and Attributs for Access Control	S
DIS 10164-10	X.742	Usage Meter Functioning	A
IS 10164-11	X.739	Metric Objects and Attributs	P
DIS 10164-12	X.745	Test Management Function	F
DIS 10164-13		Summarization Function	P
DIS 10164-14	X.737	Confidence and Diagnostic Test Categories	F
DIS 10164-15	X.746	Scheduling Function	P

**Tabelle 1: Management Function Standards (nach [SLO94])**

Obwohl diese funktionale Klassifikation für eine OSI-Umgebung entwickelt wurde, hat sie eine breite Akzeptanz bei Herstellern standardisierter und proprietärer Netzwerkmanagementsysteme erfahren. Sie kann somit als allgemeine Grundlage für die Definition der Aufgabenbereiche einer Netzwerkverwaltung angesehen werden. Auch das in Kapitel 3 beschriebene SNMP-Konzept deckt sich in Teilen mit dieser Klassifikation.

Einige Quellen, z.B. [SLO94], zählen allerdings nur das Fault Management und das Configuration Management zu den engeren Aufgabengebieten des Netzwerkmanagements. Alle fünf Funktionsbereiche werden dort dem Systems Management zugeordnet, in dem das Netzwerkmanagement nur einen kleinen Teil ausmacht.

### 2.3.1 Fault Management<sup>1</sup>

Das Fault Management beschäftigt sich mit der Erkennung, Diagnose und Behebung von Fehlern bzw. Fehlersituationen beliebiger Komponenten im Netzwerk. Dazu enthält es Funktionen, um

- Fehlerprotokolle (*Event Log* bzw. *Error Log*) zu führen,
- Fehlermeldungen zu bestätigen und darauf zu reagieren,
- Fehler zu verfolgen und zu identifizieren,
- Diagnosetests durchzuführen und
- Fehler zu korrigieren.

Der Netzwerkadministrator muß mittels des Fault Managements in der Lage sein, einen Fehler schnellstmöglich zu lokalisieren und ihn vom Rest des Netzwerkes zu isolieren, um den weiteren störungsfreien Betrieb zu sichern. Dazu rekonfiguriert bzw. ändert er das Netz, um den Einfluß auf andere Systeme so gering wie möglich zu halten. Im nächsten Schritt kann er mit der Diagnose beginnen und die fehlerhafte Komponente reparieren oder ersetzen. Mit dem Abschluß aller Arbeiten bringt der Administrator das Netzwerk wieder in seinen Ausgangszustand zurück.

Nach der Beseitigung des Fehlers und der Wiederherstellung der vollen Funktionsfähigkeit sollte das Fault Management System den Fehlerort noch eine Zeitlang überwachen. Es muß sichergestellt werden, daß das Problem wirklich gelöst und keine neuen eingeführt wurden. Eine solche Fähigkeit nennt man auch „*problem tracking and control*“.

Wie bei allen Gebieten des Netzwerkmanagements wird auch vom Fault Management erwartet, daß sein Einfluß auf Leistung und Durchsatz des Netzwerkes möglichst gering ist. Das kann u.a. durch die Nutzung von Schwellwerten erreicht werden. Erst wenn eine bestimmte überwachte Variable (z.B. die Zahl der fehlerhaft empfangenen Pakete) einen vorher festgelegten Wert überschreitet, wird ein Alarm ausgelöst. Er könnte ein entstehendes Problem des Kommunikationspfades anzeigen. Wenn der Schwellwert nicht zu groß gewählt wurde, gibt der Alarm dem Administrator Zeit und Möglichkeit, in das System einzugreifen, bevor ein größerer Ausfall eintritt. Die Basis für ein stabil arbeitendes System bleibt aber eine ausgewogene Balance zwischen Schwellwertgrenze und Netzwerkbelastung.

In der Realität sieht es jedoch häufig so aus, daß sich Fehler in komplexen Umgebungen nur schwer lokalisieren lassen. Das wird u.a. dadurch verursacht, daß einige Geräte Fehler nicht

---

<sup>1</sup> Fault - anormale Bedingung (z.B. Folge von Errors\*), die bestimmte Aktionen erfordert, um den Normalzustand wiederherzustellen

\* Error - einzelnes Fehlerereignis (z.B. Bitfehler), wird i.allg. durch implementierte Mechanismen behoben

aufzeichnen oder aber Fehler gemeldet werden, die an anderer Stelle auftreten. Häufig existieren mehrere mögliche Ursachen für ein aufgetretenes Problem, und die Vielzahl der meist sekundären Fehlermeldungen verkomplizieren die Erkennung der eigentlichen Fehlerquelle. Zum Beispiel verursacht der Ausfall der Kommunikationsleitung zwischen zwei Routern Link-Fehler in den Geräten selbst sowie Kommunikationsfehler in den Transportprotokollen der angeschlossenen Stationen. Fehler in der Kommunikationsschicht können weitere Fehler in den darauf aufbauenden, höheren Schichten der Station verursachen, bis letztendlich auch die dort laufenden Anwendungen davon betroffen sind. Falls all diese Punkte im Rahmen des Fault Managements überwacht werden, erhält man zahlreiche Meldungen, die mit dem eigentlichen Problem nichts zu tun haben und die zugrunde liegende Ursache überdecken.

### **2.3.2 Accounting Management**

Das Accounting Management ermöglicht das Erstellen von Rechnungen für die Benutzung von Ressourcen und beinhaltet Funktionen zur

- Information der Ressourcenbenutzer über Kosten,
- Tarifgestaltung im Zusammenhang mit der Benutzung von Ressourcen und
- Kostenkombination, wenn mehrere Ressourcen an der Erbringung eines Kommunikationszieles beteiligt sind.

Solche Informationen sind für viele Unternehmen interessant. Das Accounting Management ermöglicht die Sammlung und Auswertung von Daten über die Nutzung von Netzwerkdiensten für die interne Buchführung. Die verursachten Kosten werden den Abteilungen, Arbeitsgruppen und Projekten in Rechnung gestellt. Dabei geht es im allgemeinen weniger um den realen Transfer von Geldern, sondern vielmehr um eine Zuordnung des entstandenen finanziellen Aufwands zur entsprechenden Kostenstelle.

Der Netzwerkadministrator legt mittels des Accounting Managements fest, welche Arten von Informationen in den einzelnen Knoten aufgezeichnet werden. Dies könnten z.B. Angaben über den Sender und Empfänger sein, die Zeitdauer der Verbindung, die Art der Kommunikation sowie die beteiligten Applikationen bzw. Dienste. Die gesammelten Daten werden in bestimmten Intervallen an die Managementstation geschickt und dort ausgewertet. Das Resultat könnten z.B. Reports sein, die nach unterschiedlichen Kriterien aufgeschlüsselt sind.

Selbst wenn in einem Unternehmen eine Kostenabrechnung in dieser Form nicht existiert, so sind die gewonnenen Informationen für das Netzwerkmanagement sehr hilfreich. Aufgrund detaillierter Kenntnisse über die Nutzung bestimmter Ressourcen kann man Engpässe im Netz vermeiden und den zukünftigen Ausbau des Netzwerkes exakter planen.

Andererseits lassen sich mit Hilfe dieser Daten auch Rückschlüsse auf das Verhalten von Anwendern ziehen. So könnte man zum Beispiel feststellen, ob Benutzer das Netzwerk ineffektiv

verwenden und evtl. Schulungen, Unterstützungen o.ä. benötigen, oder ob Anwender ihre Zugriffsrechte mißbrauchen und Ressourcen auf Kosten anderer in Anspruch nehmen. Benutzerbezogene Informationen dieser Art müssen, soweit sie im Rahmen des Datenschutzes überhaupt zulässig sind, streng vertraulich gehandhabt werden. Der Zugriff auf diese Daten muß gesichert werden und darf nur den dafür autorisierten Personen möglich sein.

### **2.3.3 Configuration Management**

Das Configuration Management beschäftigt sich mit der Initialisierung, Wartung und Abschaltung einzelner Komponenten und logischer Subsysteme innerhalb des Netzwerkes. Dazu enthält es Funktionen, um

- Konfigurationsinformationen zu definieren,
- Parameter zu setzen, die den Routinebetrieb kontrollieren,
- Daten über den aktuellen Zustand der Systeme zu sammeln,
- Reports über den Status zu generieren und
- die Konfiguration der Systeme zu verändern.

Mittels Configuration Management ist der Netzwerkadministrator u.a. in der Lage, die Komponenten im Netzwerk zu identifizieren und ihre Charakteristik während des Startprozesses festzulegen. Dazu kann man Initial- und Defaultwerte definieren, so daß die verwalteten Ressourcen im gewünschten Modus starten, mit den korrekten Parametern versorgt werden und die geforderten Verbindungen zu anderen Netzwerkkomponenten aufnehmen. Diese Funktionalität ist insbesondere dann von Vorteil, wenn eine Netzkomponeente mehr als eine Aufgabe wahrnehmen kann, z.B. eine Arbeitsstation zusätzlich Router oder Printserver ist. Mit Hilfe vordefinierter Konfigurationsprofile kann beim Start die Betriebsart festgelegt werden.

Während des normalen Betriebs ist das Configuration Management für die Überwachung der Konfiguration und für die Ausführung von Änderungen infolge von Anwenderkommandos oder für Anforderungen aus anderen Bereichen des Netzwerkmanagements verantwortlich. So könnte zum Beispiel das Fault Management einen Fehler entdecken, der sich durch eine Konfigurationsänderung mittels des Configuration Management umgehen läßt.

Im Bereich des Configuration Managements sollten Aktionen nur durch den Netzwerkadministrator oder entsprechend autorisierte Personen durchgeführt werden können.



### **2.3.4 Performance Management**

Das Performance Management macht die Beurteilung des Verhaltens von Ressourcen sowie die Effizienz von Kommunikationsaktivitäten möglich. Es enthält Funktionen zur

- Sammlung statistischer Daten,
- Führung von Logdateien, die Auskunft über den zurückliegenden Status des Systems geben,
- Festlegung der Systemleistung unter natürlichen und künstlichen Bedingungen und zum
- Wechsel des Betriebsmodus, um Aktivitäten zum Zwecke des Performance Managements durchführen zu können.

Mit Hilfe der erfaßten Daten, deren Analyse und der Auswertung der Resultate setzt sich der Netzwerkadministrator mit Fragen nach der Auslastung, dem Durchsatz, den Antwortzeiten und evtl. vorhandenen Engpässen im Netzwerk auseinander.

Das Gebiet des Performance Managements teilt sich in zwei Kategorien auf: Monitoring und Controlling. Das Performance Monitoring überwacht die Aktivitäten im Netzwerk und das Performance Controlling ermöglicht Änderungen, um die Netzwerkleistung zu erhöhen.

Das Performance Monitoring bildet die Grundlage für das Performance Management. Häufig ist es sogar Ausgangspunkt für Managemententscheidungen im Netzwerk. Der Administrator legt einen Satz von Variablen fest, die er überwachen möchte und deren Änderungen ihm als Indikator oder Meßwert dienen. Das kann zum Beispiel die Zahl der gesendeten Bytes sein, aber auch die Menge der fehlerhaft empfangenen Pakete oder die Anzahl der Wiederholungen. Das Problem ist, die richtige Auswahl der Indikatoren zu treffen, mit denen man die Netzwerkleistung bestimmen möchte. Typische Fehler sind die Verwendung zu vieler Indikatoren, die Nutzung von Indikatoren, deren Bedeutung nicht vollständig verstanden wurde oder solcher, die von nicht allen Geräteherstellern unterstützt werden. Die meisten dieser Anzeiger eignen sich kaum für Vergleiche untereinander, und selbst bei korrekten Meßwerten bleibt immer noch die Möglichkeit der Fehlinterpretation.

Prinzipiell lassen sich Indikatoren in zwei Kategorien einteilen:

1. Dienst- bzw. Anwendungsorientiert, d.h. Angaben über Verfügbarkeit, Antwortzeiten und Präzision,
2. Effektivitätsorientiert, d.h. Angaben über Durchsatz und Auslastung bezüglich der theoretischen Ressourcenkapazität.

Da sich ein Netzwerk in erster Linie an den Anforderungen der Anwender orientiert, genießt die Gruppe der dienstorientierten Indikatoren eine höhere Priorität. Für Netzwerkmanager, die versuchen, diese Anforderungen mit möglichst niedrigen Kosten zu erfüllen, dürfte die zweite Gruppe der Indikatoren interessant sein.

### **2.3.5 Security Management**

Das Security Management unterstützt die Umsetzung von Sicherheitsrichtlinien durch Funktionen wie:

- die Erstellung, Löschung und Kontrolle von Sicherheitsdiensten und Mechanismen,
- die Verteilung sicherheitsrelevanter Informationen und
- die Meldung sicherheitsrelevanter Ereignisse.

Dabei bezieht sich das Security Management nicht auf den Einsatz oder die Verwendung bestimmter Verschlüsselungs- und Authentifizierungstechniken. Es beschäftigt sich vielmehr mit der Überwachung und Verwaltung der vorhandenen Sicherheitssysteme. Dies beinhaltet die Generierung, Verteilung und Speicherung von Verschlüsselungscodes, Paßwörtern und anderen Zugangsinformationen. Weiterhin fallen Meldungen über Versuche, die Sicherheitseinrichtungen zu überwinden, in den Bereich des Security Managements. Zu diesem Zweck werden vorrangig Logdateien und Sitzungsprotokolle geführt und aufgezeichnete Daten ausgewertet.

Die Gefahren für die Sicherheit sind in einem Netzwerk bzw. in einem verteilten System sehr vielschichtig. So kann Hardware gestohlen oder manipuliert und Software beschädigt oder gelöscht werden. Attacken dieser Art werden im allgemeinen recht schnell bemerkt. Wesentlich schwieriger zu entdecken sind dagegen passive Bedrohungen. So können Datenleitungen angezapft und die übertragenen Informationen unbemerkt aufgezeichnet werden. Da bei dieser Form des Angriffs keine Daten verändert werden, läßt er sich nur schwer nachweisen. Als probates Mittel bleiben bei solchen Gefährdungen nur präventive Maßnahmen.

Der Bereich der Systemsicherheit läßt sich in zwei Kategorien einteilen:

1. Computer Security - Schutz der Daten eines Rechners vor unberechtigten Zugriffen,
2. Network Security - Schutz der Daten während der Übertragung im Netzwerk, d.h. Schutz vor dem Abfangen, Verändern und Fälschen der Informationen.

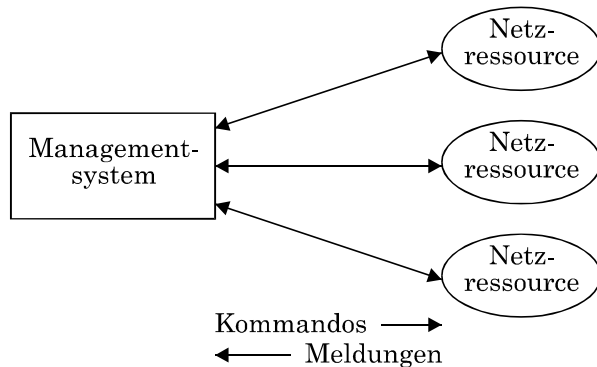
Das Security Management unterstützt beide Bereiche der Systemsicherheit für die verwalteten Ressourcen, einschließlich des Netzwerkmanagementsystems.

## **2.4 Allgemeines Modell des Netzwerkmanagements**

Dieser Abschnitt beschreibt ein allgemeines Modell, auf dem die typischen Netzwerkmanagementkonzepte (z.B. OSI CMIP oder SNMP) beruhen. Er gibt so einen Überblick über die verwendeten Begriffe und Architekturen. Ausgehend von den allgemeinen Anforderungen und Zielen eines Netzwerkmanagementsystems, die in Kapitel 2.2 beschrieben wurden, bildet die Verwaltung der Ressourcen im Netz den Kern jedes Systems. Unter Ressourcen werden dabei, wie bereits in Kapitel 2.1 erwähnt, alle Komponenten verstanden, die zur Erbringung der

Netzfunktionen beitragen, unabhängig, ob es sich um physische Ressourcen (z.B. Hosts, Router, Switches) oder logische Ressourcen (z.B. Kommunikationsverbindungen, Dienstschnittstellen) handelt.

Die Überwachung und Steuerung der Netzressourcen erfolgt durch den Austausch von Informationen zwischen Managementsystem und Ressourcen. Das Managementsystem ist ein Netzknoten, auf dem die eigentlichen Managementfunktionen ausgeführt werden (siehe dazu Abbildung 1).



**Abbildung 1: Austausch von Managementinformationen (nach [JAN93])**

Die ausgetauschten Managementinformationen lassen sich in zwei Kategorien einteilen:

1. Kommandos und
2. Meldungen.

Kommandos werden vom Managementsystem an eine Netzressource übertragen und rufen dort eine Reaktion hervor, z.B. die Aktivierung bzw. Deaktivierung einer bestimmten Komponente oder das Verändern eines Parameters. Meldungen werden spontan von der Ressource an das Managementsystem geschickt und informieren über eingetretene Ereignisse.

Wie bei jedem Datenaustausch müssen auch hier die Struktur und Reihenfolge der Managementinformationen festgelegt werden. Sie bilden die Definition für das Managementprotokoll.

Für den Transport werden die Managementinformationen in PDUs (Protocol Data Units) verpackt und an den Transportdienst der Kommunikationsschnittstelle übergeben. So ist das Managementsystem von den darunterliegenden Kommunikationsprotokollen unabhängig. In der Regel werden die PDUs mit den Managementinformationen zusammen mit allen anderen Daten in einem Netzwerk übertragen. Diese Technik wird als Inband-Management bezeichnet. In besonderen Fällen kann, z.B. aus Sicherheitsgründen, ein spezielles Netzwerk parallel zum bestehenden Datennetz aufgebaut werden. Dieses Management Network transportiert dann ausschließlich Managementinformationen. Das Verfahren nennt man Outband-Management.

### 2.4.1 Agenten

Die Übertragung von Managementinformationen und die Abwicklung des Managementprotokolls erfordern komplexe Abläufe bei der Managementstation und den Ressourcen. Da die Kapazitäten auf der Seite der Ressourcen in der Regel beschränkt sind, ist es sinnvoll, diese Funktionalität nicht von jeder Ressource zu verlangen. Deshalb benötigt das Managementsystem einen stellvertretenden Kommunikationspartner. Diese Instanz wird Agent genannt und übernimmt die Rolle des Vermittlers zwischen Manager und Ressource. Der Agent greift in einer spezifischen Art und Weise auf die Ressourcen der physischen Netzkomponente zu und übermittelt die gewonnenen Informationen (z.B. über Statusregister) per Managementprotokoll an die Managementstation. Er ist in der Regel für eine Gruppe von Ressourcen zuständig. Der Agent ist ein Softwarebestandteil der Netzkomponente (siehe Abbildung 2) und betreut alle vorhandenen Ressourcen des Gerätes (z.B. die Schnittstellen eines Routers).

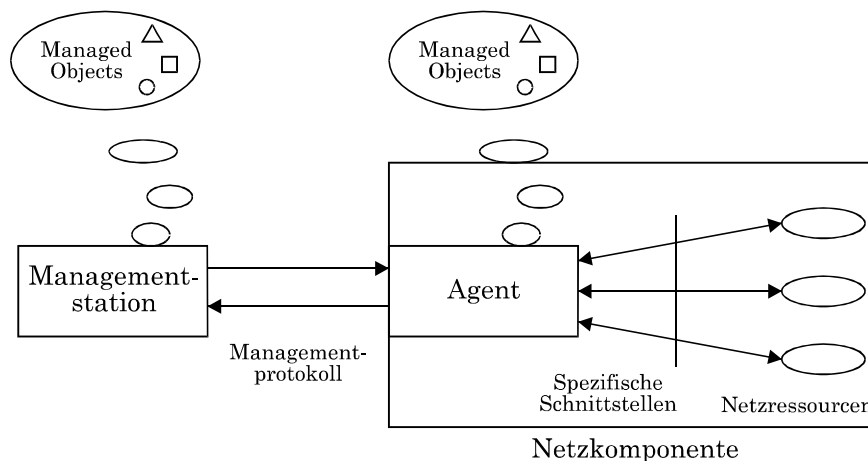


Abbildung 2: Managementstation und Agent (nach [JAN93])

### 2.4.2 Managed Objects

Eine Ressource zeichnet sich durch eine Vielzahl von Eigenschaften aus, die ihre spezielle Funktion beschreiben. Für das Netzwerkmanagement sind nur die Eigenschaften relevant, die sich mit der Netzfunktionalität beschäftigen. Somit ist nur eine Teilsicht der Ressource erforderlich. Sie wird Managed Object (Veraltetes Objekt) genannt.

Die Struktur und Eigenschaften des Managed Objects hängen stark von der jeweiligen Ressource ab. Da in einem Netzwerk sehr viele unterschiedliche Ressourcen vorhanden sind, benötigt man ein einheitliches Modell für deren Beschreibung, wobei sich nachstehender Ansatz allgemein durchgesetzt hat.

Ein Managed Object wird durch folgende Eigenschaften beschrieben:

- seine Attribute,

- die auf das Objekt anwendbaren Kommandos (Operationen),
- die Reaktionen des Objektes auf die Kommandos und
- die Meldungen, die das Objekt spontan generieren kann.

Die Attribute repräsentieren die Parameter des Objektes. Eines davon sollte der Name des Objektes sein, damit eine eindeutige Identifizierung möglich ist. Alle anderen Attribute werden auf der Grundlage vordefinierter Attributtypen beschrieben. Jedes Objekt reagiert nur auf die zugelassenen Kommandos und für jedes ist festgelegt, wie sich das Objekt verhält. Typische Anweisungen sind z.B. Attribut lesen oder Attribut ändern. In der Regel bestehen die Kommandos aus zwei Phasen:

1. Anforderung (request) und
2. Rückantwort (response).

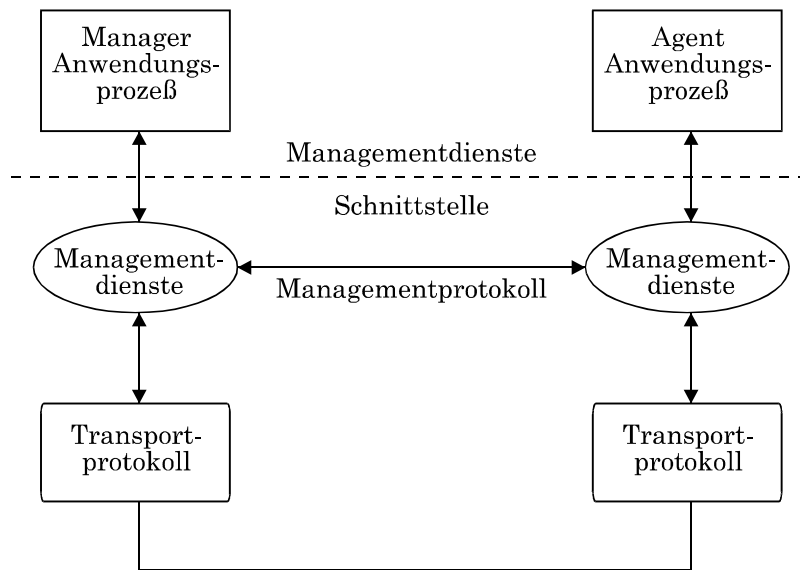
Die Rückantwort enthält eine Bestätigung bzw. Zurückweisung der Anforderung. Zusätzlich existiert i.allg. noch eine zweite Gruppe von Operationen (Meldungen), die unbestätigt ablaufen können. Sie werden bei einem besonderen oder unerwarteten Ereignis des Objektes generiert und über den Agenten an die Managementstation geschickt. Spontane Meldungen können Änderungen von Attributwerten oder Fehlerzustände anzeigen. Sie erfordern häufig eine unmittelbare Reaktion des Managementsystems.

### **2.4.3 Managementarchitektur**

Die Abbildung 3 zeigt eine generelle Architektur, auf der die meisten Netzwerkmanagementsysteme beruhen. Dabei sind Manager und Agent grundsätzlich gleich aufgebaut. Somit sind im Prinzip beide in der Lage sämtliche Managementoperationen auszuführen. Erst die konkrete Implementierung legt ihre Rolle fest.

In der obersten Schicht befinden sich die beiden Managementanwendungen (Manager und Agent). Sie rufen Funktionen des darunterliegenden Managementdienstes auf und können so Managementoperationen auf dem jeweiligen Partnersystem auslösen.

Der Managementdienst setzt die gewünschte Funktion in eine Folge von PDUs um und überträgt diese mittels des darunterliegenden Kommunikationsdienstes an sein Äquivalent im anderen System. Dort wird die Funktion aus den empfangenen PDUs dekodiert und im zugehörigen Anwendungsprozeß ausgeführt. Diese Umsetzung der Managementoperationen wird durch das Managementprotokoll definiert.



**Abbildung 3: Generelle Managementarchitektur (nach [JAN93])**

#### **2.4.4 Managementinformationen**

Unter Managementinformationen werden die in den PDUs transportierten Informationen über die Managed Objects verstanden. Sie können auf folgende Art und Weise strukturiert werden:

- als eine einfache Liste von Datenfeldern, wobei jedes Feld einen Wert speichert. Dieser Ansatz wird von SNMP verwendet.
- als eine objektorientierte Datenbank. Jedes verwaltete Element wird durch ein oder mehrere Objekte repräsentiert. Dabei enthält jedes Objekt Attribute, welches seine Eigenschaften widerspiegelt. Durch die Vererbung von Objektklassen können Beziehungen zwischen den Objekten definiert werden. Dieses Datenmodell wird vom OSI Netzwerkmanagement verwendet.
- als eine relationale Datenbank. Dabei bilden einzelne Felder in der Datenbank die Charakteristik der Netzkomponente ab. Die Struktur der Datenbank stellt die Beziehungen zwischen den Netzelementen dar.

Die Managementoperationen haben keinerlei Kenntnis über die Strukturen der Managed Objects. Sie bilden nur einen allgemeinen Rahmen und betrachten die von ihnen transportierten Objektnamen und Attribute lediglich als Anwendungsinformationen. Es ist somit die Aufgabe des Agenten zu überprüfen, ob das von einer Operation benannte Objekt überhaupt existiert bzw. ob es die benannten Attribute besitzt.

### **2.4.5 Polling vs. Traps**

Grundsätzlich existieren zwei Verfahren, um den Zustand einer Netzkomponente kontinuierlich zu überwachen: Polling und Traps.

Polling ist eine Kommunikation nach dem „request-response“-Prinzip zwischen Managementstation und Agenten. Die Managementstation kann jeden Agenten abfragen und Informationen anfordern; der Agent liefert jeweils den aktuellen Status bzw. Wert der verwalteten Objekte. Dabei kann sich eine Abfrage speziell an ein oder mehrere Objekte richten oder allgemeiner Natur sein, um das nächste passende Element zu finden. Ein Managementsystem benutzt das Polling, um Informationen über die Struktur und Konfiguration der verwalteten Systeme zu erhalten, Statusänderungen zu erfahren und die Ursachen von aufgetretenen Problemen zu ermitteln. Polling stellt relativ geringe Anforderungen an den Agenten. Das Verfahren hat allerdings auch mehrere Nachteile. Damit Polling effektiv eingesetzt werden kann, müssen die Abfragen periodisch wiederholt werden, wobei der Zyklus evtl. abhängig von der Wichtigkeit der verwalteten Objekte unterschiedlich lang konfiguriert werden kann. Dabei taucht das Problem der Rechtzeitigkeit der Benachrichtigung und der Aktualität der Informationen auf. Wird das Polling-Intervall zu groß gewählt, so kann die Reaktionszeit der Managementstation auf katastrophale Ereignisse zu lang sein. Ist das Intervall zu klein, so belastet der Managementverkehr das Netz und belegt damit Kapazität, die dem eigentlichen Datentransport nicht mehr zu Verfügung steht. Es ist leicht möglich, mit falsch gesetzten Polling-Intervallen ein ganzes Netzwerk in die Knie zu zwingen. Mit dieser Problematik hat sich u.a. Stallings in [STA96] im Kapitel „Polling Frequency“ auseinandergesetzt.

Beim ereignisgesteuerten Ansatz, auch Event Reporting oder Traps genannt, übernimmt der Agent die aktive Rolle, während die Managementstation zum Zuhörer wird und auf ankommende Nachrichten wartet. Der Agent sendet einen Trap, wenn ein vordefiniertes Ereignis (z.B. Schwellwertgrenze überschritten) oder ungewöhnliches Ereignis (z.B. irgendein Fehler) auftritt. Zusätzlich könnte der Agent auch periodisch Traps schicken, um die Managementstation über den aktuellen Zustand zu informieren. Das Senden von spontanen Meldungen hat somit den Vorteil der sofortigen Benachrichtigung. Es ist effektiver als Polling, wenn sich der Status der überwachten Objekte nur relativ selten ändert. Das Verfahren hat allerdings auch mehrere Nachteile. Es erfordert einen leistungsfähigeren Agenten sowie eine gewisse Grundintelligenz, die verhindert, daß bei nichtigen Anlässen ganze Meldungsstürme durch das Netz rauschen. Zudem muß der Trap sehr viele Informationen enthalten, die das aufgetretene Ereignis beschreiben. Deren Auswahl erfordert zusätzliche Ressourcen. Das Senden eines umfangreichen Traps macht wiederum nur Sinn, wenn eine zuverlässige Kommunikation zur Managementstation besteht und der Agent sicher sein kann, daß seine Nachricht ankommt.

Beide Varianten, Polling und Traps, sind ein anwendbares Konzept, um den Zustand der Netzkomponenten zu ermitteln, und ein NMS wird im allgemeinen auch beide Verfahren implementieren. Allerdings ist bei den verschiedenen Systemen der Schwerpunkt auf eine der beiden Methoden sehr unterschiedlich ausgeprägt. So setzen Telekommunikationsmanagementsysteme (TMN - *Telecommunication Management Network*, [SEL95]) traditionell zu einem hohen Grad auf den ereignisgesteuerten Ansatz, während SNMP den Traps nur wenig Vertrauen entgegenbringt. Das OSI-Network-Management scheint dagegen irgendwo zwischen beiden Extremen zu liegen. Einige Managementsysteme bieten daher dem Anwender die Möglichkeit, selbst den Schwerpunkt für eine der beiden Methoden festzulegen. Für welche Variante man sich letztendlich entscheidet, hängt von verschiedenen Faktoren ab, u.a.

- dem Umfang des zusätzlich generierten Netzverkehrs,
- der Robustheit in kritischen Situationen,
- der Zeitverzögerung, bis die Managementstation von einem Ereignis erfährt,
- der Nachrichtenübertragung mittels gesicherter / ungesicherter Kommunikation und
- der Zahl der zu verwalteten Geräte.



### 3 SNMP - Simple Network Management Protocol

SNMP (*Simple Network Management Protocol*) steht für ein Konzept des Netzwerkmanagements, das vorrangig in TCP/IP-Umgebungen<sup>2</sup> eingesetzt wird. Der Begriff ist leicht irreführend, da der Name des Protokolls für die Bezeichnung des gesamten Konzeptes steht. Dieses Problem teilt SNMP mit TCP/IP, das ja eigentlich „die TCP/IP-Protokoll-Familie mit ihren Anwendungen“ heißen müßte.

In einigen Quellen, z.B. Rose [ROS94], findet man für die Netzverwaltung der TCP/IP-Protokollfamilie auch die Bezeichnung *Internet-Standard Network Management Framework*.

#### 3.1 Ursprünge des TCP/IP-Netzwerkmanagements

Die Entwicklung von SNMP ist eng mit der Verbreitung und Anwendung des TCP/IP-Protokolls verknüpft. Es entstand aus der Notwendigkeit heraus, das Management im Internet (ein weltweiter Netzverbund, der aus zahlreichen Einzelnetzen besteht) effektiver zu gestalten.

Bei der Entwicklung von TCP/IP in den 70er Jahren hatte man sich kaum Gedanken um ein zukünftiges Netzwerkmanagement gemacht. Praktisch alle Rechner und Netzwerke, die man anfangs an das ARPANET, ein nordamerikanisches Forschungsnetz und Vorläufer des Internet, anschloß, wurden von Systemprogrammierern und Protokollentwicklern betreut. So konnten auftretende Managementprobleme von den zuständigen Protokollexperten gelöst werden, die über das entsprechende Hintergrundwissen verfügten.

Anfang der 80er Jahre existierte nur ein einziges Werkzeug, das für die Verwaltung IP-basierter Netze genutzt werden konnte: ICMP - *Internet Control Message Protocol*. Es ist Teil des IP-Stacks und somit auf jedem Gerät verfügbar, das IP unterstützt. ICMP stellt Mittel zur Übertragung von Kontrollnachrichten von Routern und Hosts zu einem (Monitor-) Host zur Verfügung. Auf diese Weise lassen sich Rückschlüsse über Probleme in der Systemumgebung gewinnen. Aus der Sicht des Netzwerkmanagements ist das ECHO REQUEST / ECHO REPLY-Nachrichtenpaar das nützlichste Merkmal von ICMP. Dieser Nachrichtentyp bietet einen Mechanismus zum Test der Kommunikation zwischen zwei Geräten. Der Empfänger einer ECHO REQUEST-Nachricht ist verpflichtet, umgehend den Inhalt der Nachricht mittels eines ECHO REPLY an den Sender zurückzuschicken. Ein weiteres Nachrichtenpaar namens TIMESTAMP REQUEST / TIMESTAMP REPLY funktioniert auf analoge Art und Weise und erlaubt es, die Verzögerungscharakteristik des Netzwerks zu dokumentieren.

---

<sup>2</sup> für eine detaillierte Einführung der einzelnen TCP/IP-Protokolle siehe Stevens [STE96]

Diese ICMP-Nachrichten können zusammen mit einigen IP-Headeroptionen genutzt werden, um einfache und trotzdem effektive Managementwerkzeuge zu entwickeln. Der bekannteste Vertreter ist das Programm PING (*Packet Internet Groper*). Unter Verwendung von ICMP und einigen zusätzlichen Angaben, wie der Intervalllänge zwischen den Anfragen oder der Anzahl der zu stellenden Anfragen, lassen sich mittels PING verschiedene Funktionen ausführen, z.B. kann man feststellen, ob ein Netz oder ein Gerät ansprechbar ist. Weiterhin können mit Hilfe von PING die Roundtrip-Zeiten und Datagramm-Verlustraten gemessen werden.

Die Fähigkeiten von PING und einigen zusätzlichen Tools, z.B. Traceroute, wurden anfangs den Anforderungen der Netzverwaltung gerecht. Als sich jedoch aus dem ARPANET das Internet entwickelte und die Zahl der angeschlossenen Hosts Ende der 80er Jahre exponentiell wuchs, wurde die Entwicklung eines umfangreicheren und leistungstärkeren Netzwerkmanagements notwendig.

Als Ausgangspunkt dieser Entwicklung kann SGMP (*Simple Gateway Monitoring Protocol*) angesehen werden, das im November 1987 mit dem Dokument „RFC 1028“ veröffentlicht wurde. SGMP erlaubt eine einfache Überwachung von Gateways. Da man jedoch an einer universell einsetzbaren Lösung interessiert war, wurden drei vielversprechende Vorschläge ausgearbeitet:

- HEMS (*High-Level Entity Management System*) ist eine Verallgemeinerung von HMP (*Host Monitoring Protocol*), des wahrscheinlich ersten Netzwerkmanagementprotokolls im Internet. Das Konzept fand jedoch keine praktische Umsetzung und wurde später nicht weiter verfolgt.
- SNMP (*Simple Network Management Protocol*) ist eine erweiterte Version von SGMP.
- CMOT (*CMIP over TCP*) stellt den Versuch dar, das OSI-Managementkonzept<sup>3</sup> CMIP (*Common Management Information Protocol*) für TCP/IP-basierte Umgebungen zu nutzen. Dabei werden alle Protokolle, Dienste und Datenstrukturen, die im Rahmen der ISO-Standardisierung für das Netzmanagement entwickelt wurden, übernommen und angepaßt, u.a. wird für den Transportdienst TCP statt OSI-Transport verwendet.

Anfang 1988 prüfte das IAB (*Internet Architecture Board*) die Vorschläge und beschloß, SNMP und CMOT parallel weiter zu verfolgen. Während bei SNMP das Erzielen einer kurzfristigen Lösung im Vordergrund stand, ging der Schwerpunkt bei CMOT eher in Richtung einer langfristigen und ausgewogenen Entwicklung. Man nahm an, daß in absehbarer Zeit die TCP/IP-Installationen auf OSI-konforme Protokolle umgestellt würden. Daher war geplant, beide Protokolle auf identischen Datenstrukturen zu errichten, um so einen späteren Übergang zu erleichtern. Dieser Ansatz erwies sich jedoch als nicht praktikabel. Innerhalb des OSI-Netzwerkmanagements haben Managed Objects eine komplexe Charakteristik. Sie werden als

---

<sup>3</sup> für eine Beschreibung des OSI Management Modells siehe Sloman [SLO94]

verfeinerte Entitäten mit Attributen, assoziierten Prozeduren und Nachrichten sowie objektorientierten Eigenschaften definiert. In SNMP sind solche ausgefeilten Konzepte nicht vorgesehen, es legt viel Wert auf Einfachheit. Tatsächlich handelt es sich bei den Objekten in SNMP gar nicht um Objekte im objektorientierten Sinne. Es sind vielmehr getypte Variablen mit der Möglichkeit, Zugriffsrechte wie read-only oder read-write zu definieren. Das IAB nahm von seiner ursprünglichen Forderung einer gemeinsamen Datenbasis Abstand und gestattete eine unabhängige und parallele Entwicklung beider Protokolle.

Die Tätigkeit der SNMP-Arbeitsgruppe führte im Mai 1990 zur Normung der drei Dokumente RFC 1155 (Structure of Management Information, SMI), RFC 1156 (Management Information Base, MIB-I) und RFC 1157 (Simple Network Management Protocol, SNMP). Sie erhielten den Status „recommended“ und bildeten zusammen den ersten SNMP-Standard für die Verwaltung TCP/IP-basierter Netze im Internet. Die Entwicklung von SNMP ging kontinuierlich weiter. Im März 1991 erschien die überarbeitete und erweiterte Version der Management Information Base, die MIB-II, als Norm RFC 1213. Sie ersetzte das Dokument RFC 1156, so daß bis heute die folgenden Standards die Grundlage von SNMP<sup>4</sup> bilden:

- RFC 1155 - „Structure and Identification of Management Information for TCP/IP-based Internets“ mit einer Beschreibung, wie Managed Objects in der MIB zu definieren sind.
- RFC 1213 - „Management Information Base for Network Management of TCP/IP-based Internets: MIB-II“. Hier findet sich eine Beschreibung der Managed Objects der MIB.
- RFC 1157 - „A Simple Network Management Protocol (SNMP)“ enthält schließlich die Beschreibung des Protokolls zum Management der Objekte.

Mittlerweile wird SNMP auf breiter Basis angewendet. Diese Entwicklung wurde vor allem durch das rasche Wachstum des Internets seit Anfang der 90er Jahre und der damit einhergehenden Verbreitung der TCP/IP-Protokolle unterstützt. Da die Umstellung auf OSI-konforme Protokolle ausgeblieben ist und TCP/IP auch im LAN-Bereich in stärkerem Maße eingesetzt wird, nimmt die Bedeutung von SNMP als Netzwerkmanagementkonzept weiter zu. Inzwischen wurden bereits Lösungen für die Anwendung von SNMP auf anderen Kommunikationsprotokollen (z.B. IPX, AppleTalk) erarbeitet, selbst SNMP für OSI ist möglich.

Die Hauptnachteile von SNMP in der ersten Version liegen in Mängeln bezüglich der Sicherheit und Funktionalität begründet. Deshalb wurde Oktober 1992 die „SNMPv2 Working Group“ gegründet, die sich mit der Weiterentwicklung des SNMP-Standards auseinandersetzt. Auf einige Ergebnisse dieser Arbeitsgruppe geht das Kapitel 3.7 näher ein.

---

<sup>4</sup> In dieser Arbeit ist bei der Benutzung des Begriffs SNMP implizit SNMP Version 1, kurz SNMPv1, gemeint.

## **3.2 Zielsetzung**

Das Ziel bei der Entwicklung von SNMP war die Schaffung einer allgemein anwendbaren und einfachen Lösung für das Management von Netzkomponenten. Das Hauptmerkmal liegt dabei auf Einfachheit (SIMPLE), denn Netzwerkmanagement ist (und bleibt) nur ein Zusatz, der die verwalteten Komponenten nicht von ihrer eigentlichen Tätigkeit abhalten darf. Mit anderen Worten: ein Router soll routen und ein Drucker drucken, anstatt einen Großteil seiner Kapazität für die Abwicklung von Managementprotokollen zu verwenden. Das erfordert allerdings Managementprotokolle und -operationen, die einfach aufgebaut sind, sich mit geringen Mitteln implementieren lassen und trotzdem effizient arbeiten. Marshall T. Rose beschrieb diese Grundphilosophie als „Fundamental Axiom“:

»Der Einfluß auf verwaltete Knoten durch das Hinzufügen von Netzverwaltung muß möglichst klein sein, also den kleinsten gemeinsamen Nenner darstellen.« ([ROS93])

Dieser Ansatz ist viel diskutiert und auch kritisiert worden, letztendlich hat er jedoch zum Erfolg von SNMP beigetragen.

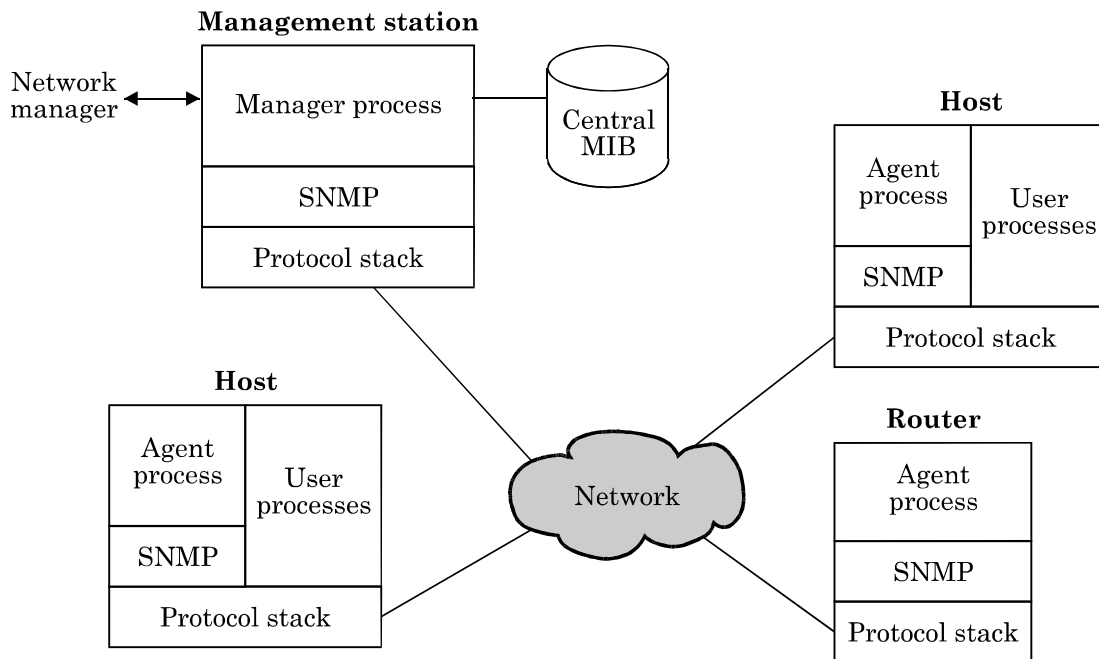
## **3.3 Grundkonzepte von SNMP**

### **3.3.1 Das Netzmodell**

Das SNMP-Modell eines verwalteten Netzes besteht aus vier Komponenten:

1. Managementstation,
2. verwaltete Knoten,
3. Managementinformationen und
4. Managementprotokoll.

Diese Elemente sind in Abbildung 4 zu sehen und werden nachfolgend erklärt.



**Abbildung 4: SNMP-Netzmodell (nach [STA96])**

Bei der Managementstation handelt es sich im Prinzip um einen „normalen“ Rechner, auf dem eine spezielle Managementsoftware läuft. Typischerweise ist es ein „Stand-alone“-Gerät, obwohl es auch als Komponente eines verteilten Systems implementiert werden kann. Die Managementstation dient als Schnittstelle zwischen dem Netzwerkadministrator (auch als Netzwerkmanager bezeichnet) und dem Netzwerkmanagementsystem. Sie umfaßt dabei zumindest

- Werkzeuge für die Datenanalyse, Fehlerbehandlung usw.,
- eine Schnittstelle, mit deren Hilfe der Netzwerkadministrator das Netz überwachen und kontrollieren kann,
- die Fähigkeit, Anfragen des Netzwerkadministrators in Anfragen an entfernte Elemente zu übersetzen sowie
- eine Datenbank, welche die Informationen aus den einzelnen MIBs der verwalteten Systeme enthält.

Allerdings sind nur die letzten beiden Punkte Teil des SNMP-Standards. In diesem Design verlagert sich die Hauptlast der Verwaltung auf die Managementstation, da SNMP die Anforderungen an die Komplexität des Managements auf seiten der verwalteten Knoten bewußt gering hält.

Die verwalteten Knoten, auch Managed Nodes genannt, können Hosts, Router, Switches, Drucker oder andere Geräte sein, die in der Lage sind, Statusinformationen der Außenwelt mitzuteilen. Um direkt von SNMP verwaltet werden zu können, muß jeder Knoten einen SNMP-Managementprozeß, einen sogenannten SNMP-Agent, ausführen. Solche Agenten sind

für die meisten Betriebssysteme der Rechner erhältlich, z.T. sind sie frei verfügbar (z.B. für diverse UNIX-Derivate), oder sie werden bereits mit dem Betriebssystem ausgeliefert. Auch die Hersteller aktiver Netzkomponenten folgen diesem Trend und implementieren Managementagenten in ihre Geräte. Jeder Agent führt eine lokale Datenbank mit Variablen, die seinen Zustand beschreiben. Er hat Zugriff auf die lokalen Ressourcen der Netzkomponente und kann sie somit beeinflussen. Der SNMP-Agent antwortet auf die Anfragen der Managementstation, führt im Auftrag der Managementstation Aktionen aus oder kann die Managementstation über außergewöhnliche Gegebenheiten und Vorkommnisse informieren.

Die Managementinformationen beschreiben detailliert, auf welche Art und Weise der Agent die Informationen über die Ressourcen einer Netzkomponente zur Verfügung stellt. Das ist notwendig, damit eine Managementstation mit diversen Komponenten kommunizieren kann. Ein großer Teil des SNMP-Modells ist die Definition dessen, was verwaltet wird und wie diese Informationen mitgeteilt werden.

Die Ressourcen einer Netzkomponente werden durch Objekte mit Attributen beschrieben. Da SNMP jedoch kein objektorientiertes System ist, besteht jedes Objekt im wesentlichen aus einer Variablen<sup>5</sup>, die eine bestimmte Eigenschaft der Ressource beschreibt. Die Menge aller derartigen Objekte werden in einer MIB (*Management Information Base*) zusammengefaßt. Die MIB fungiert so als eine Sammlung von Zugriffspunkten auf den Agenten. Die darin enthaltenen Objekte sind standardisiert und in Kategorien eingeteilt. So existiert z.B. ein Satz von Objekten für die Verwaltung verschiedener Bridges. Führt eine Managementstation eine bestimmte Überwachungsfunktion aus, so fragt sie den Wert des zugehörigen MIB-Objektes ab.

Die Verbindung zwischen der Managementstation und den Agenten wird durch das Netzwerkmanagementprotokoll SNMP hergestellt. Es bietet folgenden Basisfunktionen:

- Get: ermöglicht der Managementstation, Werte von Objekten der Agenten zu erhalten,
- Set: erlaubt es der Managementstation, Werte von Objekten der Agenten zu ändern und
- Trap: erlaubt es dem Agenten, die Managementstation über wichtige Ereignisse in Kenntnis zu setzen.

Ein Großteil der Kommunikation zwischen Managementstation und Agent folgt dem „request-response“-Prinzip.

---

<sup>5</sup> Diese Variablen werden in der SNMP-Literatur Objekte genannt. Der Begriff ist irreführend, hat sich aber trotzdem eingebürgert.

### 3.3.2 Die Protokollarchitektur

Die Übermittlung von Managementinformationen zwischen Managementstation und Agent erfolgt durch den Austausch von Protokollnachrichten. Das SNMP-Framework fordert als Transportdienst für die Protokollnachrichten lediglich einen Datagrammdienst. Da SNMP als Anwendungsprotokoll der TCP/IP-Protokollsuite entworfen wurde, setzt es auf UDP (*User Datagram Protocol*) auf. UDP bietet einen verbindungslosen Kommunikationsdienst für Anwendungen und ermöglicht so die Übertragung von Nachrichten mit einem minimalen Protokolloverhead. UDP setzt auf IP auf. Die wesentlichste Erweiterung gegenüber IP ist die Fähigkeit der Portadressierung, die durch die Angabe einer 16bit Quell- und Zielporthnummer im Header realisiert wird. Mit Hilfe dieser Ports ist eine direkte Kommunikation zwischen Anwendungen möglich. Einige dieser Portnummern, deren Werte alle unter 1024 liegen, sind für bestimmte Dienste reserviert, z.B. Port 23 für TELNET, bzw. Port 25 für SMTP (*Simple Mail Transfer Protocol*). Auch für SNMP existieren solche Standardports. So verwendet der SNMP-Agent den Port 161 für den Empfang von Nachrichten und die SNMP-Managementstation den Port 162 für den Empfang von Traps.

Sowohl die SNMP-Managementstation als auch die SNMP-Agenten müssen mindestens IP, UDP und SNMP implementieren. Die Schichtung dieser Kommunikationsprotokolle ist im Protokollkontext der Abbildung 5 zu erkennen.

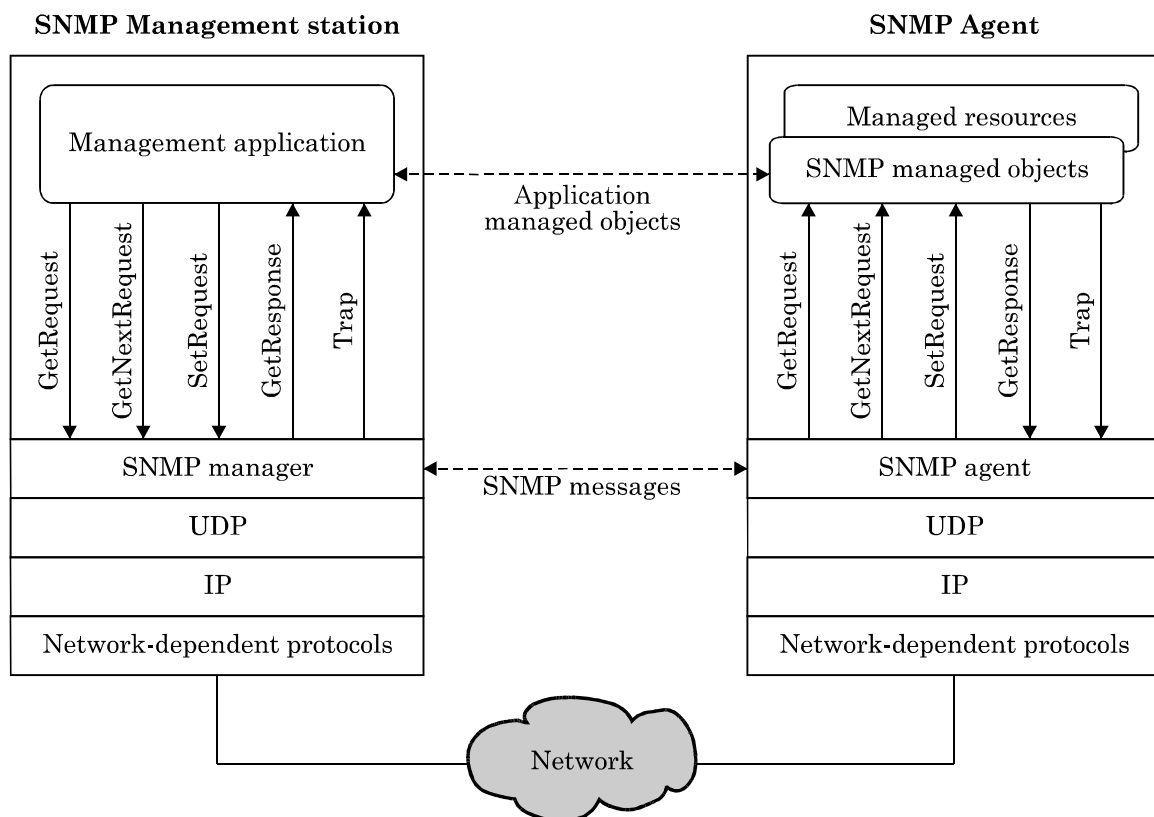


Abbildung 5: Protokollkontext von SNMP (nach [LIN96])

Die Abbildung 5 zeigt weiterhin die verschiedenen Managementoperationen. In der Managementstation stehen der Managementapplikation drei SNMP-Nachrichten zur Verfügung, die an den SNMP-Agenten geschickt werden können: GetRequest, GetNextRequest und SetRequest. Dabei sind die beiden ersten Operationen unterschiedliche Ausprägungen der Basisfunktion Get. Der Agent beantwortet alle Anfragen mit der Nachricht GetResponse, die an die Managementstation zurückgeschickt wird. Zusätzlich kann der Agent ein besonderes Ereignis mit der Operation Trap melden. Auf die Einzelheiten dieser Kommunikation geht der Abschnitt 3.5 näher ein.

Da SNMP auf UDP aufbaut, ist SNMP ein verbindungsloses Protokoll. Es existieren keine dauerhaften Verbindungen zwischen der Managementstation und den Agenten. Jede Nachrichtenübertragung stellt eine eigenständige Transaktion dar.

Die Verwendung eines Datagrammdienstes für die Übertragung von SNMP-Nachrichten ist ein viel diskutierter Kritikpunkt. Weil UDP den Empfang der Nachrichten nicht quittiert und es durchaus passieren kann, dass Datagramme verlorengehen, wird häufig die Frage gestellt, ob es nicht besser wäre, das „unreliable“ (unzuverlässige) UDP durch das verbindungsorientierte, zuverlässige TCP zu ersetzen. Eine Stellungnahme zu dieser Problematik kann man in [ROS94] von Jeff Case, einer der Mitbegründer des SNMP-Standards, finden. TCP erreicht seine Zuverlässigkeit durch wiederholte Übertragungen im Fehlerfall. Damit wird eine Applikation vor den kleinen Pannen der Kommunikation, wie z.B. plötzlicher Paketverlust, bewahrt. Gegen physische Fehler, z.B. Kabelbruch, kann auch TCP nichts ausrichten. Die implementierte Fehlerkorrektur führt in diesen Fällen nur zu langen Wartezeiten, bis das aufgetretene Problem an die Applikation zurückgemeldet wird. Ein solches Verhalten ist für ein Netzwerkmanagementsystem nicht akzeptabel. Es sollte ständig über den aktuellen Zustand des Netzes informiert sein. Umfangreiche Verzögerungen bei der Rückmeldung von Fehlern sind hier nur hinderlich und können durch den Einsatz eines verbindungslosen Protokolls, wie UDP, vermieden werden. Dann muß allerdings eine eigene Fehlerbehandlung durchgeführt werden.

### **3.3.3 Der Proxy-Agent**

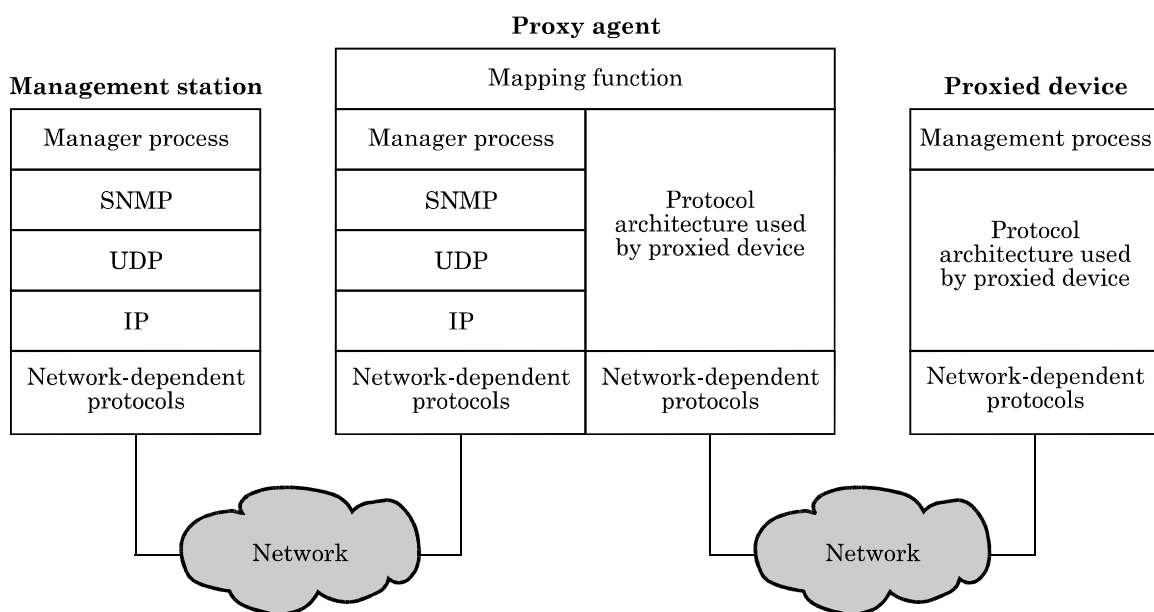
Die Anwendung von SNMP erfordert, daß alle Agenten und die Managementstation UDP und IP unterstützen. Damit werden Geräte, wie z.B. Modems, die nicht die TCP/IP-Protokollsuite unterstützen, vom Management ausgeschlossen. Zudem gibt es noch eine Vielzahl kleinerer Systeme (z.B. einfache Bridges, programmierbare Controller), die zwar über einen IP-Stack verfügen, für die es sich aber nicht lohnt SNMP zu implementieren.

Um auch Nicht-SNMP-fähige Geräte in das Management mit einbeziehen zu können, wurde das Konzept des Proxies entwickelt. Dabei agiert ein spezieller SNMP-Agent, der Proxy-Agent,



als Stellvertreter und Übersetzer für ein oder mehrere Geräte, der in der Lage ist, die SNMP-Informationen in das gerätespezifische Protokoll zu übertragen.

Die Abbildung 6 zeigt eine solche Proxy-Konfiguration. Auf der Seite der SNMP-fähigen Geräte sind die bereits aus Abbildung 5 bekannten Protokollschichten (IP, UDP und SNMP) zu sehen. Die Managementstation schickt ihre Anfragen an das Nicht-SNMP-fähige Gerät (Proxied device) über den Proxy-Agenten. Dieser übersetzt die SNMP-Anfrage mittels einer „Mapping function“ in das jeweilige Managementprotokoll des Gerätes. Wenn der Proxy eine Antwort empfängt, läuft die Konvertierung in umgekehrter Richtung ab und das Ergebnis wird an die Managementstation weitergeleitet. Dieser Mechanismus funktioniert auch beim Versand einer Ereignisbenachrichtigung (eines Traps) durch das Gerät.



**Abbildung 6: Proxy-Konfiguration (nach [SLO94])**

### 3.3.4 Trap-Directed-Polling

Die Kommunikation unter SNMP basiert auf Polling, d.h. die Managementstation fragt periodisch alle Agenten ab, um den Gesamtzustand des Netzes zu ermitteln. Zusätzlich kann der Agent bei besonderen Ereignissen einen einfachen Trap an die Managementstation schicken. Da diese Kommunikation nicht gesichert verläuft (d.h. Traps werden nicht bestätigt), können Nachrichten des Agenten verloren gehen. Traps sind in SNMP nicht zuverlässig und bilden nur einen Informationszusatz.

Falls eine Managementstation für eine große Zahl von Agenten verantwortlich ist und jeder Agent über sehr viele Objekte verfügt, stellt sich das ständige regelmäßige Abfragen aller Agenten nach verfügbaren Objektdaten schnell als unzuweckmäßig und unpraktisch heraus. Daher wurde versucht, Traps stärker in die Verwaltung mit einzubeziehen. Die Kombination

beider Verfahren hat sich im praktischen Betrieb als erstaunlich effektiv herausgestellt. Unter SNMP wird diese Technik als Trap-Directed-Polling (Trap-orientiertes Polling) bezeichnet. Dabei fragt die Managementstation die Agenten in unregelmäßigen oder zeitlich relativ großen Abständen (z.B. einmal pro Stunde) ab. Von Interesse sind bestimmte Schlüsselcharakteristika, wie beispielsweise Schnittstellenbeschreibungen oder Durchsatzstatistiken. Auf ein weiteres Polling der Agenten wird verzichtet. Statt dessen wartet die Managementstation auf Nachrichten von den Agenten über ungewöhnliche Ereignisse, z.B. Reboot eines Agenten oder Überschreiten eines Schwellwertes. Beim Empfang eines solchen Traps führt die Managementstation, zur Feststellung des aufgetauchten Problems, ein Polling beim verursachenden und den umliegenden Agenten durch.

Dieses Verfahren garantiert das schnelle Erkennen von Ausnahmesituationen, ohne daß im Agenten ein erhöhter Aufwand dafür getrieben werden muß. Der Vorteil des Trap-Directed-Polling liegt in einer substantiellen Netzentlastung und Reduktion der Agentenprozeßzeit, da die Menge der unnötig übertragenen Informationen stark verringert wird.

### **3.4 SNMP-Managementinformationen**

Den Kern des SNMP-Modells bildet eine Menge von Objekten, die von Agenten verwaltet und von der Managementstation gelesen und geschrieben werden können. Um eine Kommunikation zwischen Geräten unterschiedlicher Hersteller zu ermöglichen, müssen diese Objekte in einem einheitlichen Format definiert werden. Zusätzlich ist eine standardisierte Kodierung der Informationen während der Übertragung im Netz erforderlich, damit beide Seiten die ursprünglichen Objektdaten wieder rekonstruieren können.

Diese Problematik ist nicht neu, sondern betrifft allgemein die Datenkommunikation zwischen Rechnersystemen. Die Lösung liegt in der Anwendung formaler Datenbeschreibungssprachen, zu denen ASN.1 gehört.

#### **3.4.1 ASN.1**

ASN.1 steht für *Abstract Syntax Notation One*. Es ist eine formale Datendeklarationssprache, mit der es möglich ist, einfache Objekte zu definieren und diese anschließend zu komplexeren zusammenzufassen. In dieser Hinsicht bietet sie ähnliche Möglichkeiten wie normale Programmiersprachen für die Definition von Datenstrukturen. ASN.1 stammt von der OSI und ist unter der ISO-Norm 8824 beschrieben. Die Sprache ist recht umfangreich und weist eine OSI-typische Komplexität auf. Trotzdem wird eine Teilmenge für die Beschreibung von Managementinformationen in SNMP benutzt. Dies ist historisch bedingt, da die Managementsysteme SNMP und OSI CMOT eine gemeinsame Datenbasis verwenden sollten (siehe Abschnitt 3.1).

Folgenden Konventionen sind für ASN.1 in SNMP zu beachten: Alle Schlüsselwörter, einschließlich der einfachen, integrierten Datentypen (z.B. INTEGER), werden in Großbuchstaben geschrieben. Benutzerdefinierte Typen (z.B. TimeTicks) beginnen mit einem Großbuchstaben und müssen mindestens ein weiteres Zeichen enthalten. Bezeichner (z.B. system) beginnen mit einem Kleinbuchstaben und können ansonsten Groß- und Kleinbuchstaben, Ziffern und Bindestriche verwenden. Leerräume am Zeilenanfang haben keine Bedeutung. Alle Kommentare beginnen mit einem Doppelstrich -- und sind einzeilig.

Die Deklarationssprache ASN.1 verfügt über drei unterschiedliche Arten von Datentypen:

- *simple types* (einfache Datentypen),
- *constructed types* (zusammengesetzte Datentypen) und
- *tagged types* (gekennzeichnete Datentypen).

Die einfachen Datentypen (*simple types*) sind in Tabelle 2 zu sehen. Sie bilden die Grundlage für die Definition komplexerer Strukturen.

Datentyp	Bedeutung
NULL	Platzhalter und Anzeige, daß kein Datenwert vorhanden ist
INTEGER	Ganzzahl beliebiger Länge
OCTET STRING	Zeichenkette von 0 oder mehr Bytes
OBJECT IDENTIFIER	Objektbezeichner

**Tabelle 2: Einfache ASN.1-Datentypen**

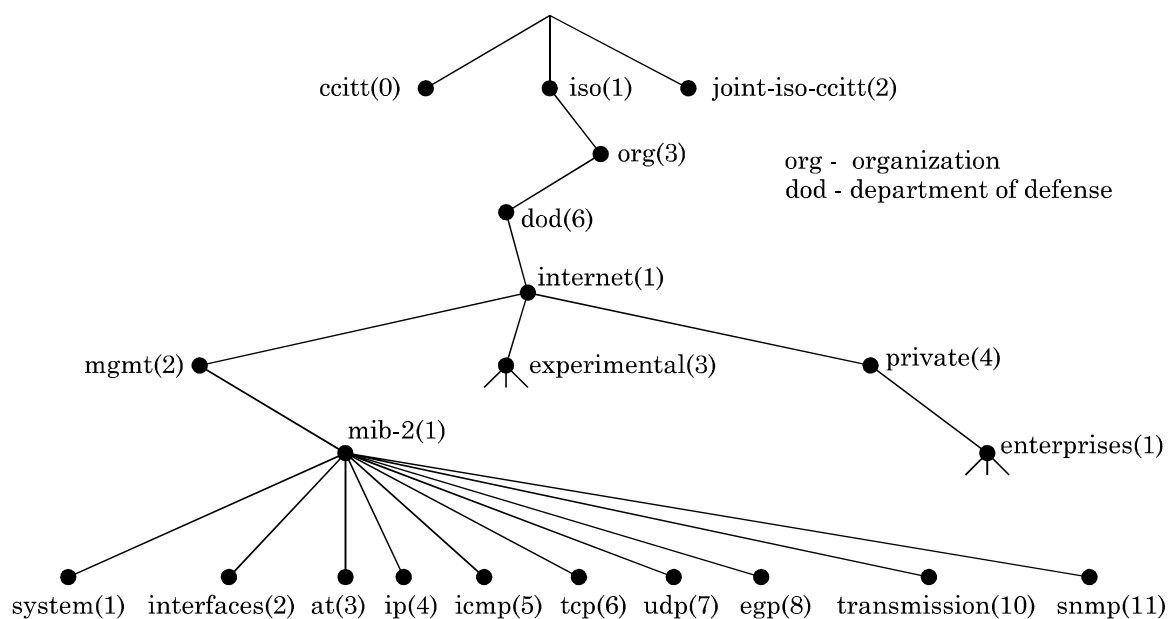
Theoretisch kann der Datentyp INTEGER in ASN.1 jeden beliebigen, ganzzahligen Wert repräsentieren. In SNMP wird die Zahlendarstellung jedoch durch die Verwendung eines 32bit Integers beschränkt. ASN.1 bietet zudem die Möglichkeit, mit der Angabe von Mengen und Wertebereichen zusätzliche Verfeinerungen der Darstellung einzuführen.

```
INTEGER (0..127)
INTEGER {up(1), down(2), unknown(3)}
```

Der OBJECT IDENTIFIER (kurz OID) wird verwendet, um Objekte zu benennen. Der benutzte Mechanismus dient der Definition eines Baumes, in dem die Objekte die Knoten des Baumes bilden. Jedem Knoten ist eine Zahl zugeordnet, die auch Sub-Identifizier genannt wird. Durch Auflistung aller Sub-Identifizier, beginnend an der Wurzel des Baumes, kann die Position jedes Objektes im Baum eindeutig bestimmt werden. Der OBJECT IDENTIFIER ist diese Liste von SubIDs, die durch Punkte getrennt werden und neben der Position des Objektes auch seinen Namen eindeutig festlegen.

Diesen Hierarchiebaum für ASN.1-Objekte zeigt die Abbildung 7. Jeder Knoten des Baumes repräsentiert ein bestimmtes Objekt, wobei die Struktur des Baumes die Gruppierung der Objekte widerspiegelt. In der obersten Ebene des Baumes befinden sich die Standardisie-

rungsorgane ISO {iso(1)} und CCITT {ccitt(0)}, sowie eine Kombination der beiden {joint-iso-ccitt(2)}. Unterhalb der ISO befindet sich ein Teilbaum für andere Organisationen {org(3)}, dem u.a. das US-Verteidigungsministerium {dod(6)} zugeordnet ist. In der Hierarchie folgt der Knoten für die Objekte des Internets {internet(1)}, der durch das IAB verwaltet wird. Der Teilbaum {mgmt(2)} enthält die standardisierten Managementinformationen, wobei der Zweig {mib-2(1)} alle mit SNMP verwalteten Objekte der gleichnamigen MIB-II repräsentiert. Die darunterliegenden Knoten {system(1), interfaces(2), ...} spiegeln die Gruppenstruktur der MIB wider (siehe dazu auch Abschnitt 3.4.4). Der Teilbaum {experimental(3)} ist für Tests vorgesehen, während der Bereich {private(4)} mit dem Unterknoten {enterprises(1)} für herstellerspezifische Erweiterungen reserviert ist.



**Abbildung 7: Hierarchiebaum für ASN.1-Objekte**

Die Bezeichnung der Knoten zeigt ein weiteres Detail. Neben der SubID kann jeder Knoten eine kurze Textbeschreibung erhalten. Hinter diesem Knotennamen wird i.allg. die SubID in runden Klammern eingetragen. Beide Varianten der Knotenbezeichnung sind bei der Angabe eines OBJECT IDENTIFIER zulässig, wobei auch Mischformen angewendet werden können. Für das bessere Verständnis ist hier ein Beispiel der möglichen OIDs für den Knoten {mgmt(2)} angegeben.

```

{iso.org.dod.internet.mgmt}
{1.3.6.1.2}
{iso.org.dod.1.2}

```

Die *constructed types* bieten in ASN.1 eine weitere Möglichkeit Datentypen zu vereinbaren. Dabei stehen die Schlüsselwörter SEQUENCE, SEQUENCE OF und CHOICE zu Verfügung. Mit Hilfe von SEQUENCE wird eine Struktur aus zusammengesetzten Datentypen definiert.

SEQUENCE OF wird für die Deklaration eines eindimensionalen Arrays eines bestimmten Typs benutzt, dessen Länge dynamisch sein kann. Durch die Kombination beider Schlüsselwörter ist das Anlegen von Tabellen in ASN.1 möglich.

```
TableEntry ::= SEQUENCE {  
    column1 INTEGER,  
    column2 OCTET STRING,  
}  
Table ::= SEQUENCE OF TableEntry
```

CHOICE beschreibt schließlich eine Vereinigung unterschiedlicher Datentypen unter einem gemeinsamen neuen. Dieses Konstrukt ist aus höheren Programmiersprachen unter dem Begriff „Union“ bekannt.

Die letzte Möglichkeit, neue Datentypen in ASN.1 zu definieren, ist die Anwendung der *tagged types*. Unter einem Tag wird eine Typkennung verstanden. Diese sind in vier Kategorien verfügbar:

- UNIVERSAL,
- APPLICATION,
- Context-specific und
- PRIVATE.

Jeder Datentyp besitzt eine solche Kennung, die aus einer Tag-Klasse und einer Tag-Nummer besteht. Die einfachen Grunddatentypen (NULL, INTEGER, ...) verwenden z.B. die Tag-Klasse UNIVERSAL. Diese Klasse ist nur für ASN.1-Sprachelemente zulässig. Durch die Benutzung eines neuen Tags kann ein bestehender Datentyp in seiner Bedeutung verändert bzw. verfeinert werden. Dies wird u.a. in der SMI (siehe Abschnitt 3.4.3) angewendet.

```
IpAddress ::= [APPLICATION 0] OCTET STRING (SIZE (4))  
Counter   ::= [APPLICATION 1] INTEGER (0..4294967295)
```

Dabei steht in eckigen Klammern die neue Tag-Klasse, gefolgt von einer Tag-Nummer. Die Tag-Klasse APPLICATION wird für die Definition von Datentypen in MIB-Modulen verwendet. Die benutzte Tag-Nummer muß innerhalb des Moduls eindeutig sein. Die Tag-Klasse PRIVATE steht für herstellerspezifische Typkennungen zur Verfügung.

Neben den Datentypendeklarationen beinhaltet ASN.1 auch einen umfangreichen Makromechanismus. Ein Makro kann als eine Art Prototyp verwendet werden, um eine Menge neuer Typen und Werte mit einer eigenen Syntax zu definieren. Aufgrund der Komplexität kann allerdings nicht auf Einzelheiten eingegangen werden. Beschreibungen zu dieser Problematik findet man zum Beispiel in [STA96].

SNMP macht lediglich an einer Stelle von Makros Gebrauch. Innerhalb der SMI werden zwei Makros definiert: OBJECT-TYPE für die Definition verwalteter Objekte und TRAP-TYPE für die Definition von Traps. Alle weiteren Möglichkeiten von Makros kommen in SNMP nicht zur Anwendung.

### 3.4.2 ASN.1-Transfersyntax

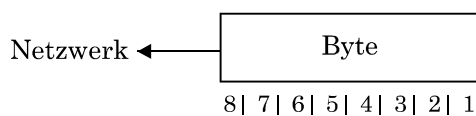
Die ASN.1-Transfersyntax definiert, wie Werte von ASN.1-Datentypen für eine Übertragung eindeutig in eine Folge von Bytes kodiert und am anderen Ende wieder dekodiert werden. Die von ASN.1 verwendete Transfersyntax heißt BER (*Basic Encoding Rules*). Sie ist gemeinsam mit ASN.1 von der OSI entwickelt und unter der ISO-Norm 8825 beschrieben worden. BER ist relativ komplex, da es für eine platzsparende Kodierung optimiert wurde. Aus heutiger Sicht wäre eventuell ein einfacheres Schema ohne übermäßig sparsamen Umgang mit Speicheranforderungen günstiger gewesen. Aufgrund des Umfangs von BER können hier nicht alle Details erläutert werden. Eine genauere Beschreibung kann man z.B. in [ROS94] finden.

Die Kodierregeln basieren auf einem einfachen Prinzip. Jede Variable wird durch drei Größen kodiert:

- *Tag* (Kennzeichen),
- *Length* (Länge) und
- *Value* (Wert),

weshalb dieses Verfahren auch TLV-Kodierung heißt. Dabei beschreibt *Tag* den Datentyp, mit dem alle Variablen eindeutig gekennzeichnet sind. *Length* gibt die Länge des nachfolgenden Value-Feldes an und *Value* selbst beinhaltet den eigentlichen Wert der Variablen. Diese Regeln werden bei zusammengesetzten Variablen mehrfach angewendet. So können strukturierte Objekte auf eine Verkettung von einfachen Variablen reduziert werden.

Damit die Informationen zwischen verschiedenen Maschinenarchitekturen übertragen werden können, legt BER eine Bitordnung fest. Das MSB (*most significant bit*) ist das Bit 8 und das LSB (*least significant bit*) Bit 1. Dabei wird das hochwertigste Bit bei einer Übertragung zuerst übermittelt (siehe Abbildung 8).

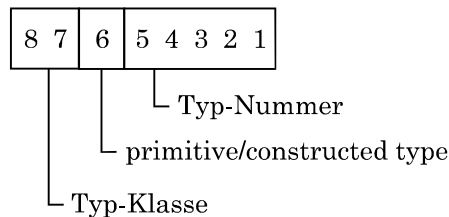


**Abbildung 8: Bitordnung nach BER**

Die eigentlichen Daten werden nun wie folgt kodiert:

Das Tag-Feld ist in drei Abschnitte unterteilt (siehe Abbildung 9) und enthält die Tag-Klasse und die Tag-Nummer des Datentyps. Die Bits 8 und 7 kennzeichnen die Tag-Klasse. Die Werte 00, 01, 10 und 11 stehen für UNIVERSAL, APPLICATION, Context-specific und PRIVATE. Das Bit 6 zeigt an, ob es sich bei der Variable um einen einfachen Typ (Bit 6 = 0) oder um einen zusammengesetzten Typ (Bit 6 = 1) handelt. Die restlichen fünf Bits kodieren die Tag-Nummer, die jedem Datentyp zugeordnet ist (z.B. Tag-Nummer 2 für INTEGER). Die Werte von 0 bis 30 reichen bisher aus, um alle in SNMP vorkommenden Typen zu kennzeichnen.

Sollten einmal weitere Kennungen benötigt werden, so kann mit dem reserviertem Wert 31 angezeigt werden, daß die Tag-Nummer im nächsten bzw. in den nächsten Bytes folgt.



**Abbildung 9: Tag-Feld nach BER**

Dem Tag-Feld folgt das Length-Feld. Es gibt an, wieviele Bytes die eigentlichen Daten belegen. Längen von weniger als 128 Byte werden direkt in einem Byte kodiert, dessen MSB auf 0 gesetzt ist. Größere Längenangaben erfordern mehrere Bytes. Dabei wird im ersten Byte das MSB auf 1 gesetzt und die restlichen sieben Bit zeigen an, wieviele Bytes noch folgen, in denen der eigentliche Längenwert steht. Mit dieser Methode läßt sich eine Zahl von maximal 126 Bytes Länge<sup>6</sup> kodieren. Bei acht Bits pro Byte ergeben sich  $126 * 8 = 1008$  Bits und somit einen Wert von  $2^{1008} - 1$ . Diese Zahl, die nur die Länge des nachfolgen Value-Feldes angibt, ist größer als der Adreßraum aller derzeitigen Rechner und bietet somit hinreichend Platz für zukünftige Anwendungen.

Im Anschluß an das Length-Feld wird der Wert der Variablen im Value-Feld kodiert. Die jeweilige Kodierung hängt vom vorhandenen Datentyp ab. Am einfachsten ist der Typ NULL. Es werden keine Werte übertragen und das Length-Feld ist auf 0 gesetzt. Auch der OCTET STRING bereitet keine Probleme. Die Bytefolge wird einfach von links nach rechts übertragen. Ganzzahlen (z.B. INTEGER) werden im Zweierkomplement kodiert. Dabei werden führende Nullen weggelassen. Somit erfordert eine positive Ganzzahl unter 128 ein Byte und eine positive Ganzzahl unter 32768 zwei Byte. Die Übertragung erfolgt in der Network-Byteorder bzw. „Little Endian“-Format, d.h. das höherwertige Byte wird zuerst übermittelt. Etwas umfangreicher ist die Kodierung eines OBJECT IDENTIFIER. Er wird in seine Sub-Identifizier zerlegt und diese als eine Folge von Bytes kodiert. Dabei werden die beiden ersten SubIDs aus Platzgründen zusammengefaßt. Dies ist möglich, da nach ISO-Definition die erste Zahl immer im Bereich {0,1,2} liegt und die zweite kleiner als 40 ist. Die beiden Stellen werden daher mit dem Wert  $40a + b$  kodiert. Um die einzelnen Sub-Identifizier voneinander unterscheiden zu können, wird jeweils das MSB auf 0 gesetzt. Damit müssen allerdings SubIDs größer 127 in mehreren Bytes kodiert werden. Eine solche Folge ist an einem gesetzten MSB zu erkennen, wobei das Ende durch ein MSB mit dem Wert 0 gekennzeichnet ist.

Die Abbildung 10 zeigt einige Beispiele für die Kodierung von ASN.1-Datentypen mittels BER.

---

<sup>6</sup> Der Wert 127 ist für zukünftige Erweiterungen reserviert.

	Tag			Length	Value			
INTEGER (101)	00	0	02	0 0 0 0 0 0 0 1	0 1 1 0 0 1 0 1			
NULL	00	0	05	0 0 0 0 0 0 0 0				
OID {1.3.6.1}	00	0	06	0 0 0 0 0 0 1 1	0 0 1 0 1 0 1 1	0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 1	

**Abbildung 10: Beispiele für BER-Kodierungen**

### 3.4.3 SMI

SMI steht für *Structure and Identification of Management Information* (Struktur und Identifizierung von Managementinformationen) und ist der Titel des Dokuments RFC 1155. Es enthält die Definition der Datenstrukturen von SNMP und beschreibt das Verfahren, wie Managed Objects in SNMP abgebildet werden. Dazu werden folgende Vereinbarungen getroffen:

- Der Zugriff auf Managed Objects erfolgt über eine virtuelle Datenbasis, die *Management Information Base* bzw. kurz MIB (siehe Abschnitt 3.4.4) genannt wird. Dort sind alle vorkommenden Objekte in der Datendeklarationssprache ASN.1 beschrieben.
- Jedes Objekt in der MIB ist über seinen Namen und seine Syntax definiert.
- Der Name eines Objektes wird durch einen OBJECT IDENTIFIER festgelegt, dessen Wert eindeutig ist.
- Die Syntax beschreibt die Datenstruktur eines Objektes. Dafür können alle in SNMP zulässigen ASN.1-Datentypen verwendet werden.

Um die Möglichkeiten für die Definition von Objekten zu erweitern, führt die SMI sechs neue Datentypen ein, die in Tabelle 3 aufgelistet sind. Die Definitionen der neuen Datentypen beruht auf der Verfeinerung bereits bekannter Datentypen. Dazu wird die Methode des Taggings eines ASN.1-Datentyps benutzt, die im Abschnitt 3.4.1 (*tagged types*) erläutert wurde.

Durch die Benutzung eines neuen Tags wird ein existierender Datentyp in seiner Bedeutung geändert. Eine solche Definition in formaler ASN.1-Syntax ist hier für den neuen Datentyp `IpAddress` angegeben:

```
IpAddress ::= [APPLICATION 0] OCTET STRING (SIZE (4))
```

Die Tabelle 3 gibt die Definitionen in vereinfachter Form wieder. Aufgelistet sind die neuen Datentypen (Spalte 1), die zugrunde liegenden Basistypen (Spalte 2) und ihre neue Bedeutung (Spalte 3).



Datentyp	verwendeter Typ	Bedeutung
IpAddress	OCTET STRING	dezimale Internetadresse (OCTET STRING hat feste Länge von 4 Byte)
NetworkAddress	CHOICE	allgemeiner Datentyp für die Netzadresse, derzeit jedoch nur IP-Adresse definiert
Counter	INTEGER	32bit Zähler, beginnt nach Überschreiten des Maximalwertes wieder bei 0
Gauge	INTEGER	32bit Pegelanzeige, Wert kann steigen und fallen, überschreitet den Maximalwert jedoch nicht
TimeTicks	INTEGER	32bit Zeitzähler in 1/100 sec
Opaque	OCTET STRING	Verpackungstyp für beliebige, nicht genormte Formate

**Tabelle 3: Verfeinerte Datentypen in der SMI**

Um ein Managed Object konkret zu definieren, wird in der SMI auf den Makromechanismus von ASN.1 zurückgegriffen. In streng formaler ASN.1-Syntax wird das Makro OBJECT-TYPE eingeführt, das vier notwendige Parameter (SYNTAX, ACCESS, STATUS, DESCRIPTION) und weitere optionale Parameter hat. Mit ihnen können die Merkmale der zu verwaltenden Variable festgelegt werden.

SYNTAX definiert den Datentyp der Variablen, wobei alle zulässigen Typen der SMI verwendet werden dürfen. Zusätzlich kann der Wertebereich bei INTEGERn und abgeleiteten Datentypen sowie die Länge bei OCTET STRINGs eingeschränkt werden.

ACCESS legt den möglichen Zugriff auf die Objektvariable fest. Die zulässigen Werte sind *read-only* (nur lesen), *read-write* (lesen / schreiben), *write-only* (nur schreiben) und *not-accessible* (nicht zugreifbar). Hat eine Variable das Attribut *read-only*, so kann sie von der Managementstation nur gelesen und nicht verändert werden.

STATUS legt die Anforderungen an die Implementierung der Objekte fest. So bedeutet der Wert *mandatory* (vorgeschrieben), daß verwaltete Knoten dieses Objekt implementieren müssen, während es bei *optional* freiwillig ist. Der Status *obsolete* (veraltet) zeigt an, daß das Objekt nicht mehr verwendet werden soll und der Wert *deprecated* (mißbilligt) dient als Vorwarnung. Es solches Objekt wird in einer der nächsten MIB-Versionen den Status *obsolete* erhalten und zukünftig nicht mehr in der MIB vorhanden sein.

DESCRIPTION ist für den Netzwerkadministrator gedacht. Das Attribut beschreibt in einem kurzen Text die Funktion und das Verhalten der Objektvariable. Häufig finden sich hier Hinweise für den sinnvollen Gebrauch.

Ein einfaches Beispiel (Abbildung 11) einer Deklaration mittels des Makros OBJECT-TYPE soll die Anwendung der aufgeführten Attribute verdeutlichen. Die Variable 'sysDescr' enthält eine Beschreibung des jeweiligen Gerätes. Die Zuweisung '::=' in der letzten Zeile ordnet der Objektvariablen den OBJECT IDENTIFIER {system 1} zu und legt so die Position des Objektes im Hierarchiebaum der MIB eindeutig fest.

```
sysDescr OBJECT-TYPE
    SYNTAX  OCTET STRING (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A textual description of the entity.  This value
        should include the full name and version
        identification of the system's hardware type,
        software operating-system, and networking
        software.  It is mandatory that this only contain
        printable ASCII characters."

    ::= {system 1}
```

**Abbildung 11: Beispiel einer Objektdeklaration**

### 3.4.4 MIB

Die MIB (*Management Information Base*) ist eine virtuelle Datenbasis aller Managed Objects in SNMP, die formal in der Datendeklarationssprache ASN.1 beschrieben sind. Das Attribut virtuell wird benutzt, da keine Angaben über eine konkrete Implementierung gemacht werden. Typisch für SNMP ist jedoch die Verwendung einfacher Variablenlisten.

Die MIB residiert in der Managementstation und ist (zumindest in Ausschnitten) auch auf allen Agenten vorhanden. Sie muß zwischen Managementstation und Agent übereinstimmen, damit eine vollständige Verwaltung der Netzkomponente möglich ist. Objekte des Agenten, von denen die Managementstation keine Kenntnis hat, können in das Management nicht mit einbezogen werden. Die MIB des Agenten ist (z.B. bei Routern, Switches) in die Firmware des Gerätes integriert. Sie ändert sich nur mit einer neuen Version des Gerätes. Die MIB in der Software des Managementsystems kann i.allg. durch den Netzwerkadministrator erweitert bzw. geändert werden. Auf diese Weise können die besonderen Eigenschaften eines Gerätes in das Managementsystem übernommen werden.

Wie bereits im vorherigen Kapitel beschrieben, werden alle Managed Objects mittels des Makros OBJECT-TYPE in der MIB abgebildet. Dort werden die Objekte, die im Zusammenhang stehen, zu Gruppen zusammengefaßt und diese Gruppen in Modulen gesammelt. Das bekannteste Modul ist im Dokument RFC 1213 beschrieben und umfaßt alle Objekte, die ein SNMP-Agent standardmäßig unterstützen sollte. Es wird auch Standard-MIB RFC 1213 bzw. MIB-II genannt. Leider werden die Begriffe „MIB-Modul“ und „MIB“ in verschiedenen Quellen synonym verwendet. Der Begriff „MIB-Modul“ stellt aber die exaktere Bezeichnung dar.

Ein MIB-Modul weist eine festgelegte Struktur auf:

```
<moduleName> DEFINITIONS ::= BEGIN
    <linkage>
    <declarations>
END
```

Die Beschreibung eines Moduls beginnt mit der Festlegung des Modulnamens <moduleName>, wobei die nachfolgenden Definitionen durch das führende Schlüsselwort BEGIN eingeleitet werden. Das Modul läßt sich in die beiden Abschnitte <linkage> und <declarations> unterteilen. In <linkage> werden Definitionen aus anderen Modulen importiert. Dies können Objektgruppen, Datentypen oder Makros sein. Für die MIB-II sieht das zum Beispiel so aus:

```
IMPORTS mgmt, NetworkAddress, IpAddress, Counter, Gauge,
        TimeTicks FROM RFC1155-SMI
        OBJECT-TYPE FROM RFC-1212;
```

In <declarations> sind die eigentlichen Definitionen des Moduls enthalten. Diese lassen sich grob in folgende Abschnitte unterteilen:

- ASN.1-Spracherweiterungen, z.B. DisplayString ::= OCTET STRING,
- Gruppendefinitionen, z.B. system OBJECT IDENTIFIER ::= { mib-2 1 },
- Managed Objects und
- TRAP-Definitionen.

Die Beschreibung eines Moduls schließt mit dem Schlüsselwort END. Weitere Informationen zum Aufbau eines MIB-Moduls sind in [PER93] „Understanding SNMP MIBs“ zu finden.

Die Objekte in einem MIB-Modul werden anhand ihres OBJECT IDENTIFIERS identifiziert (siehe Abschnitt 3.4.1). Die OIDs aller Module bilden einen gemeinsamen Hierarchiebaum, in dem die Objekte der MIB die Knoten des Baums bilden. Die MIB-II wird z.B. unter dem Knoten {iso.org.dod.internet.mgmt.mib-2 bzw. 1.3.6.1.2.1} abgebildet. Damit dieser Mechanismus funktioniert, ist es notwendig, daß alle OIDs eindeutig sind. Das ist für die standardisierten MIB-Module<sup>7</sup> gegeben.

Auch die Hersteller von Netzkomponenten können den Hierarchiebaum der MIB erweitern. Dies gibt ihnen die Möglichkeit, spezielle Eigenschaften ihrer Geräte, die durch die Standard-MIB RFC 1213 nicht repräsentiert werden, mit eigenen Objektdefinitionen abzubilden. Diese herstellerspezifischen MIB-Module werden „enterprise-specific MIB“ oder „private MIB“ genannt. Um Abbildungskonflikte bezüglich der OIDs zwischen den Herstellern zu vermeiden, wurde der Teilbaum {iso.org.dod.internet.private.enterprises bzw. 1.3.6.1.4.1} eingerichtet. Hier kann jeder Hersteller einen Knoten beantragen, der ihm durch die IANA (*Internet Assigned Numbers Authority*) zugewiesen wird. So hat zum Beispiel die Firma Proteon den Knoten {enterprises.proteon(1)} und die IBM {enterprises.ibm(6)}. Innerhalb des eigenen Teil-

---

<sup>7</sup> Neben der MIB-II (RFC 1213) gibt es weitere öffentliche MIB-Module, z.B. RFC 1493 Bridge-MIB, RFC 1514 Host Resources MIB, die jedoch zum Teil den Status „draft“ haben oder von experimentellem Charakter sind.

baums muß allerdings jeder Hersteller selbst für die Konsistenz bei der Objektnummernvergabe sorgen.

Als Beispiel für die verschiedenen MIB-Module soll hier kurz die MIB-II RFC 1213 vorgestellt werden. Es ist bereits die zweite Version der sogenannten Standard-MIB, in der alle Objekte aufgelistet sind, die ein verwalteter Knoten, auf dem SNMP läuft, unterstützen soll. Jedes vorhandene Objekt wurde nach Kriterien ausgewählt, die u.a. in [ROS93] beschrieben sind:

- Das Objekt muß entweder für die Fehleranalyse oder das Feststellen der Konfiguration wichtig sein.
- Das Objekt muß genügend allgemein gehalten sein, damit es auf vielen unterschiedlichen Plattformen gefunden werden kann.
- Das Objekt darf sich nicht mittels einfachen Operationen aus anderen Objekte ableiten lassen.
- Aufgrund der unzureichenden Zugriffskontrolle darf das Objekt keine bzw. nur begrenzte sicherheitsrelevante Eigenschaften haben.

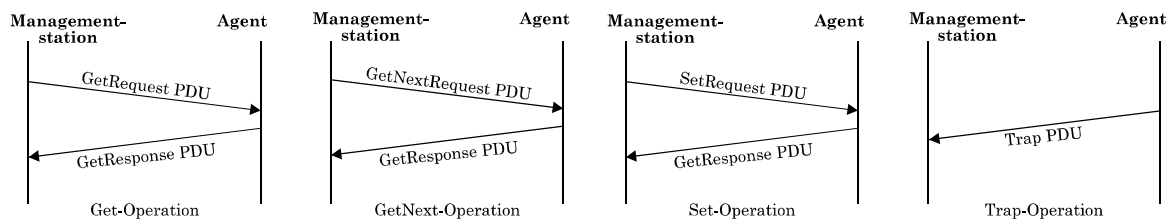
Entstanden ist eine Sammlung von 175 Objekten, die sich in 10 unterschiedliche Gruppen unterteilen. Die Tabelle 4 bietet einen kurzen Überblick über die MIB-II. Sie listet in der ersten Spalte die verschiedenen Gruppen der MIB-II auf und gibt in der zweiten Spalte die Anzahl der verwalteten Objekte in der jeweiligen Kategorie an. Zusätzlich sind in der dritten Spalte einige knappe Informationen zu den Eigenschaften der Gruppe aufgeführt. Auf eine ausführliche Beschreibung aller Objekte muß verzichtet werden. Sie würde den Rahmen dieser Arbeit sprengen. Weiterführende Informationen sind im RFC 1213 sowie in [STE96], [STA96] und [JAN93] zu finden.

Gruppe	Objektanzahl	Bedeutung
system	7	Name, Standort und Beschreibung der Netzkomponente
interfaces	23	Konfigurationsdaten und Statistiken über alle physischen Netzwerkschnittstellen
at	3	„address translation“; Umsetzung von Protokolladressen in Netzadressen; inzwischen aber veraltet Problem: Bindung mehrerer Protokolle an ein Netzinterface
ip	42	IP-Protokoll, enthält u.a. Routing-Tabelle
icmp	26	ICMP-Protokoll
tcp	19	TCP-Protokoll, enthält u.a. Tabelle über aktuell existierende TCP-Verbindungen
udp	6	UDP-Protokoll; statistische Informationen über empfangene und gesendete Datagramme
egp	20	Daten über das Routing-Protokoll EGP
transmission	0	bisher noch reserviert
snmp	29	Informationen zu SNMP, z.B. fehlerhafte Communities

**Tabelle 4: Objektgruppen der MIB-II**

### 3.5 Protokoll und Operationen

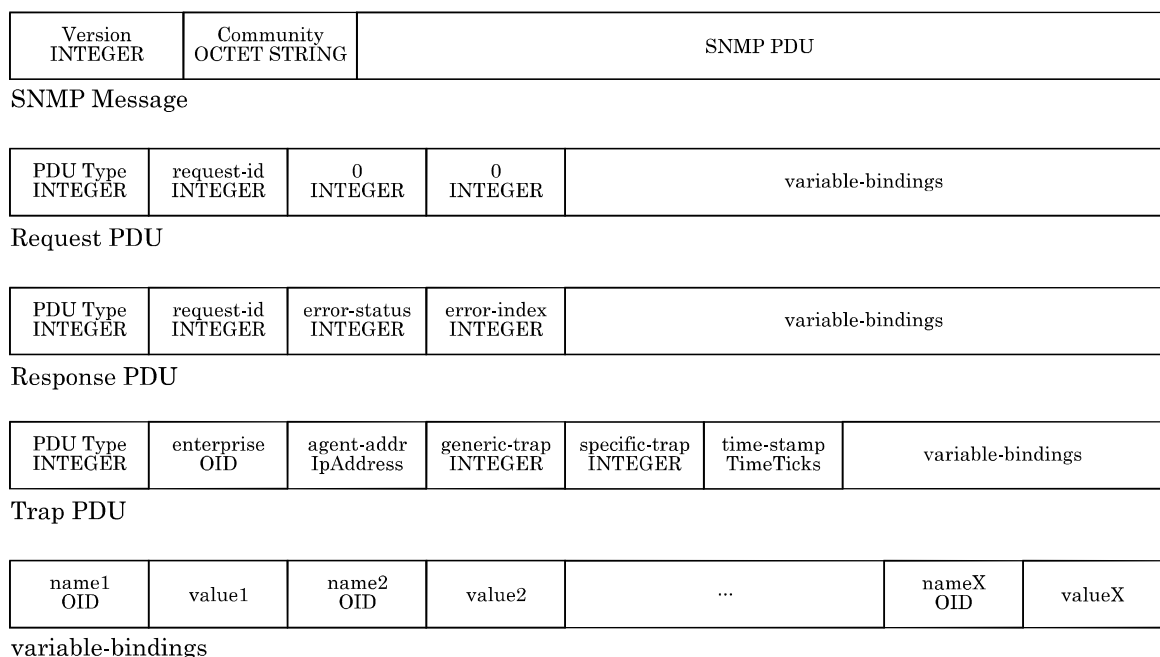
Das SNMP-Protokoll ist ein sogenanntes Request-Response-Protokoll, wobei SNMP-Nachrichten zwischen der Managementstation und den Agenten ausgetauscht werden. Die Nachrichten werden spontan geschickt. Es bedarf somit keiner speziellen Vorbereitung (z.B. Verbindungsaufbau) zwischen den Kommunikationspartnern. Auf jede Anfrage (*request*) erfolgt, mit Ausnahme der Trap-Operation, eine Rückantwort (*response*). Dabei müssen die Anfragen nicht synchronisiert werden. Es ist also möglich, mehrere Requests hintereinander zu senden, ohne zwischendurch auf die Rückantworten zu warten. Die Beziehung zwischen Request und Response wird über ein spezielles Feld (request-id) in der SNMP-Nachricht hergestellt. Es dient gleichzeitig der Erkennung von verlorenen oder verdoppelten Nachrichten. Da das Protokoll ungesichert abläuft, ist es Aufgabe der Managementstation, bei einer nicht angekommenen Nachricht entsprechende Maßnahmen zu ergreifen. Für das SNMP-Protokoll wurden vier Operationen definiert: Get, GetNext, Set und Trap. Der Kommunikationsablauf dieser Operationen zwischen Managementstation und Agent ist in Abbildung 12 ersichtlich. Die Details der Operationen werden in den nachfolgenden Abschnitten erläutert.



**Abbildung 12: Ablauf der SNMP-Operationen**

Die SNMP-Nachrichten sind in einer formalen Beschreibung in ASN.1-Syntax definiert. Für eine Übertragung werden die Strukturen mittels der Transfersyntax von ASN.1 kodiert. Somit sind SNMP-Nachrichten im Gegensatz zu anderen Protokollen (z.B. IP, TCP, ...) nicht an feste Bytegrenzen gebunden. Die Länge und relative Position der einzelnen Felder wird dynamisch während der Kodierung festgelegt. Es existiert jedoch die Vereinbarung, daß die Agenten der verwalteten Knoten nur Nachrichten bis zu einer Gesamtlänge von 484 Bytes verarbeiten müssen. Größere Nachrichten können sie mit einer Fehlermeldung abweisen. Trotzdem wird empfohlen, auch größere Nachrichtenlängen zu unterstützen.

In Abbildung 13 ist der Aufbau der SNMP-Nachrichten (*SNMP Messages*) zu sehen. Jede von ihnen besteht aus einem Header, der eine Versionsnummer (*Version*) und die Community enthält sowie der eigentlichen SNMP PDU (*Protocol Data Unit*). Diese PDU ist für die Operationen GetRequest, GetNextRequest und SetRequest (siehe Request PDU), inklusive der Rückantwort GetResponse (siehe Response PDU), nach einer gleichen Grundstruktur aufgebaut. Nur die PDU für die Trap-Operation weicht von diesem Format ab. Sie wird im Abschnitt 3.5.5 näher erläutert.



**Abbildung 13: SNMP-Nachrichtenformate (nach [STA96])**

Die Bedeutung der einzelnen Felder in Abbildung 13 ist in Tabelle 5 ersichtlich.

Feld	Bedeutung
PDU-Type	Kennung der PDU; abhängig von der jeweiligen Operation
request-id	Identifizier, an dem die zu einem Request zugehörige Response-PDU ermittelt werden kann
error-status	Status der ausgeführten Operation; kann die Werte noError(0), tooBig(1), noSuchName(2), badValue(3), readOnly(4) und genErr(5) annehmen
error-index	Index der Variablen in variable-bindings, die den von error-status gemeldeten Fehler verursacht hat
variable-bindings	Liste von Objektvariablen, bestehend aus einer OID und einem Wert, der bei GetRequest und GetNextRequest auf NULL gesetzt wird, um die Konsistenz der Struktur zu erhalten

**Tabelle 5: Felddescriptions der SNMP-Nachrichten**

### 3.5.1 GetRequest

Die Operation GetRequest (PDU-Type 0) fragt die Werte von einer oder mehreren Objektinstanzen ab. Dazu führt die zugehörige PDU die Liste *variable-bindings*, die pro Objektinstanz ein OID-Feld und ein Feld für den abgefragten Wert vorsieht. Beim Request werden jedoch alle Wertfelder auf NULL gesetzt. Auch die Felder *error-status* und *error-index* werden initialisiert. Sie erhalten den Integerwert 0. Jede GetRequest-Operation wird mit einer GetResponse-PDU beantwortet.

Ein Wort zu den hier aufgeführten Objektinstanzen: Im Gegensatz zu den bisher besprochenen Objekten, wird unter einer Objektinstanz eine reale Ausprägung eines Objektes verstanden, die eine bestimmte Variable in einem verwalteten Knoten repräsentiert. Die Unterscheidung von Objekt und Objektinstanz ist notwendig, da es von einem Objekt mehrere Instanzen geben kann. Dieser Fall tritt bei Tabellenspalten auf.

Da das SNMP-Protokoll ausschließlich mit Objektinstanzen arbeitet, muß die Benennung derselben eindeutig geregelt sein. Sehr einfach ist das für Objekte, von denen es prinzipiell immer nur eine einzige Instanz gibt und die somit keine Tabellenspalte darstellen. Dann wird an die OID des Objektes der Sub-Identifizier 0 angehängt, um die zugehörige Objektinstanz zu erhalten. Dies soll am Beispiel des Objektes {system.sysDescr} verdeutlicht werden.

```
system.sysDescr.0
1.3.6.1.2.1.1.1.0
```

Bei Objekten, von denen mehrere Instanzen existieren können, ist die Bestimmung der OID etwas aufwendiger. Da in SNMP keine kompletten Tabellen- oder Zeilenobjekte verändert werden können, muß es sich um das Objekt einer Tabellenspalte handeln. Nun ist die Selektion einer bestimmten Zeile der Tabelle notwendig, um die Objektinstanz eindeutig festzulegen. Dazu wurden Indexspalten in der Tabelle eingeführt, deren Wert als Endung an die OID des abzufragenden Spaltenobjektes angehängt wird. Die Objektbeschreibung der Tabelle in der MIB enthält die Information, welche Spalte(n) als Index dienen.

Der Sachverhalt soll an einem Beispiel verdeutlicht werden. Die Instanzen der Spalte der Tabelle {interfaces.ifTable} werden durch die Benutzung des Wertes der Spalte (ifEntry.ifIndex) ausgewählt. Die Instanz der Spalte {ifEntry.ifDescr} ergibt nun für das erste Interface den Wert:

```
interfaces.ifTable.ifEntry.ifDescr.1  
1.3.6.1.2.1.2.2.1.2.1
```

### 3.5.2 GetResponse

Die PDU GetResponse (PDU-Type 2) wird bei den Operationen GetRequest, GetNextRequest und SetRequest zurückgegeben. Sie enthält die Werte der angeforderten Objektinstanzen in der gleichen Reihenfolge wie beim vorangegangenen Request oder eine entsprechende Fehlermeldung.

Bei einem Fehler wird eine Kopie der Liste *variable-bindings* aus dem Request zurückgegeben und die Felder *error-status* und *error-index* gesetzt. Zum Beispiel wird der Status „noSuchName“ zurückgeliefert, wenn eine unbekannte Objektinstanz angesprochen wurde. In diesem Fall enthält das Feld *error-index* die Angaben für die Position der verursachenden Objektvariable.

### 3.5.3 GetNextRequest

Die Operation GetNextRequest ist eine spezielle Variante der Operation GetRequest. Dabei wird jedoch nicht die angesprochene Objektinstanz zurückgeliefert, sondern die nächste vorhandene. Dies ist aufgrund der Namensgebung der Objektinstanzen möglich. Es läßt sich eine lexikographische Ordnung angeben, so daß für zwei Instanznamen a und b immer eine der folgenden Bedingungen gilt:  $a < b$ ,  $a = b$  oder  $a > b$ . Somit wird bei einem GetNextRequest immer der Objektname und Objektwert derjenigen Instanz zurückgegeben, die in lexikographischer Ordnung unmittelbar der abgefragten Instanz folgt.

Dieses Verhalten ist in einigen Situationen äußerst hilfreich. So gestattet der GetNextRequest einen Test, ob der Agent eine bestimmte Objektvariable implementiert. Weiterhin kann durch wiederholtes Ausführen der Operation GetNextRequest die komplette Datenbasis des Agenten durchlaufen werden. Dieses Vorgehen erlaubt der Managementstation ein dynamisches Er-



kunden der MIB-Struktur des Agenten. Der häufigste Anwendungsfall für die Operation `GetNextRequest` dürfte jedoch bei der Abfrage von Tabellen zu finden sein. Aufgrund der Namensgebung in Tabellen ist bei wiederholten Aufrufen von `GetNextRequest` ein Durchlaufen der Spalten möglich, d.h. für eine bestimmte Spalte werden nacheinander die Zeilenwerte zurückgegeben bis das Ende der Tabelle erreicht ist. Dies ermöglicht auch bei Tabellen mit unbekannter Dimension relativ einfache Abfragen.

#### 3.5.4 SetRequest

Mittels der Operation `SetRequest` kann die Managementstation die Änderung einer oder mehrerer Objektinstanzen veranlassen, deren OID und zukünftiger Wert in der Liste *variable-bindings* übertragen werden. Die Operation ist atomar, d.h. entweder werden alle spezifizierten Instanzen geändert oder keine davon. Verläuft die Operation erfolgreich, so liefert der Agent die Liste *variable-bindings* mit den entsprechend gesetzten Werten zurück. Kann wenigstens eine Variable nicht gesetzt werden, so wird eine Kopie der Liste aus dem `SetRequest` an die Managementstation zurückgeliefert. In diesem Fall kennzeichnet das Feld *error-status* den Fehlergrund und das Feld *error-index* die verursachende Variable.

#### 3.5.5 Trap

Die Trap-PDU wird von einem Agenten spontan im Falle eines ungewöhnlichen Ereignisses an die Managementstation gesendet. Dabei werden in SNMP sieben Ereignisse ausgewertet:

- *coldStart* (Neustart des Agenten),
- *warmStart* (Reinitialisierung des Agenten),
- *linkDown* (Schnittstelle in den Status „DOWN“ gewechselt),
- *linkUp* (Schnittstelle in den Status „UP“ gewechselt),
- *authenticationFailure* (Authentifizierung einer PDU fehlgeschlagen),
- *egpNeighborLoss* (EGP-Nachbar des Agenten in den Status „DOWN“ gewechselt) und
- *enterpriseSpecific* (herstellerspezifisches Ereignis),

wobei letzterer für herstellerdefinierte Erweiterungen zur Verfügung steht. Im Gegensatz zu allen anderen Operationen ist der Trap eine unbestätigte Operation, d.h. es wird keine Antwort der empfangenen Station erwartet. Da das Format der Trap-PDU von den anderen abweicht, sind die Felder aus Abbildung 13 in Tabelle 6 erläutert.

<b>Feld</b>	<b>Bedeutung</b>
enterprise	Wert des Objektes sysObjectID
agent-addr	IP-Adresse des Agenten, der den Trap erzeugt hat
generic-trap	beschreibt das Trap-Ereignis; kann die Werte coldStart(0), warmStart(1), linkDown(2), linkUp(3), authenticationFailure(4), egpNeighborLoss(5) und enterpriseSpecific(6) annehmen
specific-trap	enthält bei einem enterpriseSpecific-Trap eine nähere Beschreibung, ansonsten 0
time-stamp	Wert des Objektes sysUpTime zum Zeitpunkt des Ereignisses
variable-bindings	Liste von Objektvariablen; liefert weitere Informationen zu Trap

**Tabelle 6: Ergänzende Feldbeschreibung der Trap-PDU**

### 3.6 SNMP Security

Netzwerkmanagement kann als eine Form der verteilten Anwendung gesehen werden; im Fall von SNMP bestehend aus einer Managementstation und mehreren Managed Nodes. Da man mittels Managementprotokoll jeden Knoten beeinflussen kann, sollte der Zugriff nur berechtigten Entitäten möglich sein. Es wird ein Sicherheitsmodell benötigt, wobei SNMP die Bereiche Authentifizierung und Zugriffskontrolle abdeckt.

- Authentifizierung - Sie beschränkt den Zugriff auf die lokale Datenbasis des Agenten auf autorisierte Managementstationen.
- Zugriffskontrolle - Sie ermöglicht unterschiedliche Zugriffsrechte für verschiedene Managementstationen.

SNMP in der Version 1 benutzt, gemäß der Dokumentation RFC 1157, ein sehr einfaches Sicherheitssystem, das sogenannte Community-Konzept. Eine SNMP-Community definiert eine Beziehung zwischen dem Agenten und der Managementstation. Die Communities werden lokal in den Agenten verwaltet. Jede Community besteht aus einem Community-Namen<sup>8</sup> sowie aus einer Menge von Adressen von Managementstationen, die Teil dieser Community sind. Bei jeder Operation muß nun die jeweilige Managementstation den Community-Namen mit angeben, um sich als gültiges Mitglied einer Community auszuweisen.

---

<sup>8</sup> In einigen Dokumentationen wird der Begriff „Community“ synonym für „Community-Name“ verwendet.

### 3.6.1 Authentifizierung

Eine Authentifizierung stellt sicher, daß eine Kommunikation zwischen zwei Einheiten zulässig ist. Im Falle von SNMP bedeutet dies, daß der Empfänger die Nachricht einer bestimmten Quelle zuordnen kann.

Die Definition im RFC 1157 legt für SNMP eine sehr einfache Authentifizierung fest. Jede Nachricht, unabhängig von der eigentlich auszuführenden Operation, enthält einen Community-Namen. Dieser Name fungiert als Paßwort. Somit wird die Nachricht als authentisch angesehen, wenn der Absender das korrekte Paßwort übermittelt. Der Community-Name wird dabei unverschlüsselt, d.h. im Klartext, übertragen. Er besteht i.allg. aus ASCII-Zeichen, deren Groß-/Kleinschreibung zu beachten ist. Die Defaulteinstellung vieler Agenten ist „public“.

Aufgrund der einfachen Authentifizierung, die bei jeder Operation notwendig ist, bleibt allerdings die Gefahr der unberechtigten Manipulation durch das Abfangen und Verändern einer Nachricht. Sie kann durch den Einsatz unterschiedlicher Communities für GET und SET Operationen, zusammen mit einem eingeschränkten Adreßraum, für den die jeweilige Community gültig ist, verringert werden. Ganz ausschließen lassen sich solche Attacken jedoch nicht. Daher wurde diese Schwäche bei der Festlegung der Objekte für die Standard-MIB [RFC1213] berücksichtigt. Es wurden nur Elemente implementiert, die keinen bzw. lediglich geringen Einfluß auf die Sicherheit eines Gerätes haben.

### 3.6.2 Zugriffsberechtigung

Mit der Definition einer Community kann der Agent den Zugriff auf seine Datenbasis (seine lokale MIB) beschränken. Durch die Verwendung mehrerer Communities ist es dem Agenten möglich, den verschiedenen Managementstationen eine unterschiedliche Ansicht auf seine MIB zu geben. Diese Beschränkung wird durch zwei Eigenschaften definiert:

1. MIB View - Ein View (*Ansicht*) ist eine Teilmenge von Objekten der MIB. Für jede Community kann ein anderer View festgelegt werden. Die Objektmenge eines Views kann beliebig gewählt werden und muß sich nicht auf einen bestimmten Teilbaum der MIB beschränken.
2. Access Mode - Für jede Community wird ein Access Mode (*Zugriffsart*) definiert, der einen der beiden Werte READ-ONLY (*nur lesen*) bzw. READ-WRITE (*lesen / schreiben*) annehmen kann.

Die Kombination von MIB View und Access Mode wird als SNMP Community Profile (*Gruppendefinition*) bezeichnet. Somit besteht ein solches Profile aus einer Teilmenge der MIB sowie einer Zugriffsberechtigung auf die darin enthaltenen Objekte. Ist z.B. READ-ONLY definiert, so kann jede Managementstation, die diese Community benutzt, nur Lesezugriffe auf Objekte ausführen.

Es ist jedoch zu beachten, daß es zwei verschiedene Zugriffsbeschränkungen gibt. Zum einen den Access Mode, der im Community Profile definiert wird sowie das ACCESS-Attribut, das bei jedem MIB-Objekt angegeben wird. Erst die Kombination beider Attribute legt die zulässigen Operationen fest. Diesen Sachverhalt verdeutlicht die Tabelle 7. Dort sind die möglichen Attributwerte des Access Mode in der Community denen des ACCESS-Attributs im Managed Object gegenübergestellt. In den resultierenden Zellen der Tabelle erkennt man, welche Operationen für das jeweilige Paar von Attributwerten ausführbar sind. Einen Sonderfall stellt das Paar {READ-WRITE; WRITE-ONLY} dar. Prinzipiell läßt diese Kombination nur die Operation SET zu. In Abhängigkeit von der Implementierung sind bei einigen Agenten trotzdem Lesezugriffe auf ein solches Objekt möglich.

Community Access Mode	Object ACCESS-Attribut			
	READ-ONLY	READ-WRITE	WRITE-ONLY	Not Accessible
READ-ONLY	3	3	1	1
READ-WRITE	3	2	4	1

	erlaubte Operationen
1	keine
2	GET, GET-NEXT, SET, TRAP
3	GET, GET-NEXT, TRAP
4	SET; implementierungsabhängig auch GET, GET-NEXT, TRAP

**Tabelle 7: Beziehung zwischen Access Mode und ACCESS-Attribut (nach [ROS93])**

### 3.7 Weiterentwicklungen des SNMPv1-Standards

SNMP in der ersten Version ist nicht frei von Mängeln. Der Standard, der als kurzfristige und schnelle Lösung für Netzwerkmanagementprobleme geschaffen wurde, weist im praktischen Einsatz verschiedene funktionale Beschränkungen auf. Einige der häufig genannten Kritikpunkte sind folgende:

- SNMP eignet sich nicht für die Erfassung größerer Datenmengen. Die Abfrage ganzer Tabellen (z.B. der Routing-Table) stellt sich als sehr aufwendiger Prozeß dar.
- Traps werden in SNMP nicht bestätigt. Durch die Verwendung des verbindungslosen UDP-Protokolls ist der Versand von Nachrichten in kritischen Situationen problematisch. Der Agent weiß nicht, ob seine Meldung die Managementstation erreicht hat.

- Die Kommunikation zwischen Managementstationen wird nicht unterstützt. So ist der Aufbau von hierarchischen bzw. verteilten Managerstrukturen nicht möglich. Damit wird die Verwaltung sehr großer Netze erschwert.
- Das Sicherheitsmodell von SNMP ist mangelhaft und bietet keinen zuverlässigen Schutz vor unberechtigten Zugriffen. Aufgrund der einfachen Authentifizierung können Community-Namen durch das Abhören bzw. Abfangen von Nachrichten ermittelt werden. Mit diesen Informationen können die Netzkomponenten durch unberechtigte Dritte manipuliert werden. Daher haben einige Hersteller die SET-Operation nicht in ihre Geräte implementiert. SNMP eignet sich somit eher für die Überwachung als für die Steuerung von Netzen.

Insbesondere die unzureichenden Sicherheitsmechanismen waren ausschlaggebend für die Weiterentwicklung von SNMP. Im Oktober 1992 begann die Entwicklung des neuen Standards, genannt SNMP Version 2 bzw. kurz SNMPv2. Der bestehende Standard erhielt den Namen SNMPv1. Es wurden zwei Arbeitsgruppen gebildet, von denen sich die erste nur mit Fragen zur Sicherheit von SNMPv2 befaßte, während die zweite sich mit allen anderen Problemen des Managementprotokolls auseinandersetzte. Bereits im März 1993 legten beide Gruppen ihre Vorschläge vor, die unter den Dokumenten RFC 1441 bis RFC 1452 veröffentlicht wurden.

### **3.7.1 SNMPv2p**

Allein die Steigerung von ehemals drei RFCs auf nunmehr zwölf zeigt den Umfang des neuen Managementprotokolls SNMPv2. Daher kann hier verständlicherweise nicht auf alle Einzelheiten eingegangen werden. Mehr Details und eine ausführliche Beschreibung von SNMPv2 kann man z.B. in [JAN93] und [STA96] finden. Die wesentlichen Änderungen lassen sich allerdings in folgende Kategorien einteilen:

- erweiterte Managementinformationen,
- neue bzw. geänderte Protokolloperationen und
- ein neues Sicherheitsmodell.

Die SMI (*Structure of Managementinformation*) von SNMPv2 wurde gegenüber der bisherigen SMI [RFC1155] in verschiedener Hinsicht erweitert. Die meisten Korrekturen betreffen Änderungen von Details mit dem Ziel, Unstimmigkeiten bzw. Ungenauigkeiten zu beseitigen. Bemerkenswert ist die Erweiterung des Macros OBJECT-TYPE um vier zusätzliche Datentypen (siehe Tabelle 8), die Definition der zwei neuen Macros MODULE-IDENTIFY und NOTIFICATION-TYPE sowie die Einführung des neuen MIB-Teilbaums „snmpV2“.

<b>Datentyp</b>	<b>Bedeutung</b>
UInteger32	vorzeichenloser INTEGER-Typ
Counter64	64 Bit langer Zählertyp
BIT STRING	Zeichenkette mit 0 oder mehr Bits
NSAP	Datentyp für OSI-Netzadressen (NsapAddress)

**Tabelle 8: Zusätzliche SNMPv2-Datentypen**

Auch das eigentliche Managementprotokoll von SNMPv2 wurde einer Überarbeitung unterzogen. Herausgekommen sind zwei neue Operationen und verschiedene Detailänderungen. Die neue GETBULK-Operation ermöglicht eine vollständige Abfrage von Tabellen mit lediglich einer REQUEST-Anforderung der Managementstation und gestattet so eine effizientere Übertragung großer Datenmengen. Dabei ist die GETBULK-Operation konzeptionell nicht neu. Sie basiert auf einer wiederholt ausgeführten GETNEXT-Operation. Während allerdings mit GETNEXT maximal eine Zeile der Tabelle abgefragt werden kann und die Managementstation wiederholte Anforderungen über das Netz schicken muß, wird bei GETBULK die Wiederholung im Agenten ausgeführt. Der schickt eine einzige große Rückantwort (RESPONSE) an die Managementstation. Das Ergebnis ist eine Reduktion der Netzlast durch das Zusammenfassen der Einzelabfragen.

Die zweite neue Operation ist der INFORM-REQUEST. Sie dient der Übertragung von Managementinformationen von einer Managementstation zu einer zweiten. Mit dieser Operation ist der Aufbau hierarchischer Managementstrukturen möglich. Lokale Manager überwachen die untergeordneten Agenten und melden besondere Ereignisse mittels eines INFORM-REQUEST (ähnlich wie bei einem Trap) an die übergeordnete Managementstation. Diese bestätigt allerdings den Empfang mittels eines INFORM-RESPONSE.

Zu den überarbeiteten Punkten des Managementprotokolls SNMPv2 gehören u.a. die Traps. Die Kritik bezog sich auf ihren zu geringen Standardinformationsgehalt, da weiterführende Angaben von der jeweiligen Implementierung abhängig waren. In SNMPv2 sind nun zwei Variablen festgelegt, die in jedem Trap enthalten sein müssen. Weiterhin wurde auch die Zahl der Fehlercodes in den RESPONSE-Nachrichten erweitert. Sie stieg von bisher sechs auf neunzehn Meldungen, von denen insbesondere die SET-Operation profitiert. Die möglichen Ursachen für das Scheitern einer solchen Operation werden jetzt wesentlich besser differenziert.

Die Spezifikation zu SNMPv2 enthält auch eine neue Definition der zulässigen Transportdienste. Während für SNMPv1 nur UDP spezifiziert war, läßt SNMPv2 nun folgende Protokollumsetzungen zu:

- User Datagram Protocol (UDP),
- OSI Connectionless-Mode Network Service (CLNS),
- OSI Connection-Oriented Network Service (CONS),
- Novell Internetwork Packet Exchange (IPX) und
- AppleTalk.

Dabei wird allerdings das Transportprotokoll UDP als favorisierte Umsetzung für SNMP hervorgehoben.

Einer der ständigen Kritikpunkte von SNMPv1 war das unzureichende Sicherheitsmodell. Deshalb wurde für SNMPv2 ein völlig neues Sicherheitskonzept entworfen, das vier Ziele definiert:

- die Nachprüfbarkeit des Nachrichtenursprungs (*origin authentication*),
- die Nachrichtenintegrität (*message integrity*),
- der Schutz vor Wiederholungen (*replay protection*) und
- die Vertraulichkeit einer Nachricht (*privacy*).

Um diese Ziele zu erreichen, benutzt SNMPv2 zwei Mittel: Prüfsummen und Verschlüsselung. Dabei kommt der MD5-Algorithmus zum Einsatz, der zu jeder Managementnachricht einen elektronischen Fingerabdruck in Form einer 128 Bit langen Prüfsumme berechnet. Diese wird zusätzlich übertragen und stellt so die Integrität der Nachricht sicher. Da einer der Parameter des MD5-Algorithmus eine Zeitmarke ist, kann die Laufzeit einer Nachricht kontrolliert werden. Dies bietet Schutz vor Wiederholungen; die Nachricht ist nur innerhalb eines kleinen Zeitintervalls gültig. Das Verfahren erfordert allerdings synchronisierte Uhren zwischen Sender und Empfänger. Die Vertraulichkeit einer Nachricht sichert SNMPv2 mittels Verschlüsselung. Dazu wird der DES-Algorithmus mit einer Schlüssellänge von 128 Bit verwendet. Da nur berechtigte Sender den geheimen Schlüssel kennen, kann gleichzeitig der Ursprung einer Nachricht überprüft werden. Nachteilig ist, daß DES zur starken Kryptographie zählt und daher gewissen Exportbeschränkungen der USA unterliegt.

Das Sicherheitskonzept von SNMPv2 ist in einen Kommunikationsrahmen eingebettet, der über das bisherige Gruppenmodell (Communities) hinausgeht. Die Beziehung zwischen Managementstation und Agent wird durch die Einführung der SNMPv2 Party und des SNMPv2 Context wesentlich erweitert. Eine SNMPv2 Party ist eine virtuelle Ausführungsumgebung für SNMP-Operationen, in der eine definierte Menge von Parametern (z.B. Zugriffsrechte, Schlüssel) die Randbedingungen für die Generierung und Verarbeitung von SNMP-Nachrichten festlegt. Unter SNMPv2 Context ist die Objektansicht auf ein oder mehrere MIBs zu verstehen.

Der Kommunikationsrahmen hat den speziellen Namen dieser Version von SNMPv2 geprägt. Der Ansatz wird party-based SNMPv2 bzw. kurz SNMPv2p genannt. Diese Variante von SNMPv2 konnte sich jedoch nicht durchsetzen. Im Bereich der Sicherheit sprengten die Defi-

nitionen das Grundprinzip der Einfachheit. Der Vorschlag wurde als zu komplex bewertet und erhielt später den Status „historic“ (veraltet).

### **3.7.2 SNMPv2c**

Aufgrund des Scheiterns von SNMPv2p wurde Anfang 1996 erneut eine Arbeitsgruppe gebildet. Ihre Aufgabe war die Überarbeitung der bisherigen Spezifikationen. Dabei wurde das kritisierte Sicherheitsmodell von SNMPv2p entfernt und wieder auf das Community-Konzept von SNMPv1 zurückgegriffen. Alle anderen Neuerungen von SNMPv2 wurden übernommen und in einem neuen Vorschlag unter den Dokumenten RFC 1901 bis RFC 1908 veröffentlicht. Er erhielt den Namen community-based SNMPv2 bzw. kurz SNMPv2c.

SNMPv2c ist bis jetzt kein offizieller Managementstandard. Er wird immer noch unter dem Status „experimental“ (experimentell) geführt. Trotzdem hat er bei den Geräteherstellern eine gewisse Akzeptanz erreicht und wird von einigen kommerziellen Produkten unterstützt.

### **3.7.3 SNMPv2u**

Parallel zu den Arbeiten an SNMPv2c fanden weitere Entwicklungen statt. Eine dieser SNMPv2-Varianten ist user-based SNMPv2 bzw. kurz SNMPv2u. Der Vorschlag setzt auf den RFCs 1902 bis 1908 auf und führt ein neues, anwenderbasiertes (user-based) Sicherheitsmodell ein, das unter den RFCs 1909 und 1910 dokumentiert ist. SNMPv2u hat den Status „experimental“ und wird bisher kaum angewendet.

### **3.7.4 SNMPv3**

Keine der SNMPv2-Varianten<sup>9</sup> hat sich auf breiter Basis durchsetzen können. Nur das Managementprotokoll SNMPv1 wird allgemein akzeptiert und unterstützt. Um diese verfahrenre Situation zu bereinigen, wurde 1997 die SNMPv3-Arbeitsgruppe ins Leben gerufen. Sie soll die noch ausstehenden Probleme lösen, mit dem Ziel, wieder einen einheitlichen SNMP-Standard zu schaffen. Dabei einigte man sich zu Beginn der Aktivitäten darauf, die vorhandenen Definitionen soweit wie möglich zu nutzen. Da die RFCs zu SNMPv3 vermutlich erst 1998 veröffentlicht werden, sind Informationen zum zukünftigen Standard noch sehr rar.

SNMPv3 ist modular aufgebaut und seine Architektur läßt sich flexibel erweitern. Es wird u.a. mehrere Sicherheitssysteme enthalten, die parallel eingebunden werden können. Spezifiziert ist bisher das „User Based Security Model“, ähnlich wie bei SNMPv2u.

---

<sup>9</sup> Leider geht kaum eine Literaturquelle auf die unterschiedlichen SNMPv2-Varianten ein. In den meisten Dokumentationen zu diesem Thema geht man davon aus, daß SNMPv2 bereits Standard ist bzw. in nächster Zeit Standard wird.



## 4 Java

Java ist ein Produkt der Firma „Sun Microsystems“, das im Mai 1995 auf der SunWorld '95 der Öffentlichkeit vorgestellt wurde. Seitdem hat das Konzept einer leistungsfähigen, objektorientierten und plattformunabhängigen Programmiersprache, deren Applikationen in das World Wide Web (siehe Abschnitt 4.2) eingebunden werden können, für viel Furore gesorgt.

Dieses Kapitel soll einen Einblick in das Konzept von Java geben, da diese Programmiersprache für die Entwicklung der Anwendung „JReport“ genutzt wurde. Es ist allerdings keine Einführung in die Programmierung mit Java, da in der Fachliteratur umfangreiche Anleitungen zu diesem Thema zu finden sind, z.B. [CAM96], [ECK97] und [WEB96].

### 4.1 Die Entwicklung von Java

Verschiedene Publikationen vermitteln bei einem ersten und schnellen Überblick über Java den Eindruck, daß die Sprache speziell für den Einsatz im Internet entwickelt wurde. Dabei ist das Konzept von Java wesentlich älter und existierte bereits, bevor jemand ernsthaft an das World Wide Web dachte.

Anfang 1991 stellte die Firma „Sun Microsystems“ ein Team unter der Führung von James Gosling und Patrick Naughton zusammen, mit dem Ziel, Basistechnologien für den Zukunftsmarkt „Consumer Electronics“ (z.B. Set-Top-Boxen für den TV-Bereich) zu entwickeln. Unter dem Namen „Green Project“ sollte ein kleines, objektorientiertes System entworfen werden, das an keine Hardwareplattform gebunden ist und somit flexibel bezüglich Hardwareausstattung und Prozessortyp sei. James Gosling beschrieb in einem Interview der „SunWorld Online“ die Aufgaben folgendermaßen:

„The goal was ... to build a system that would let us do a large, distributed, heterogeneous network of consumer electronic devices all talking to each other.“

Das Ergebnis nach ca. zweijähriger Entwicklung war ein Gesamtkonzept aus einem netzwerktauglichem Betriebssystem und einer C++-ähnlichen Programmiersprache namens „Oak“ (*Object Application Kernel*). Dabei war Oak vollständig plattformunabhängig konzipiert, um den Anforderungen der zahlreichen Hersteller in der Elektronikbranche zu entsprechen. Dies wurde durch die Benutzung eines architekturneutralen Objektdatenformates erreicht. Oak-Programme wurden in einen Zwischencode (den sogenannten Bytecode) übersetzt, der zur Laufzeit auf einer festgelegten virtuellen Maschine interpretiert und ausgeführt wurde.

Trotz der richtungsweisenden Entwicklungen war dem Gesamtkonzept kein Erfolg beschieden. Der Markt der „intelligenten Unterhaltungselektronik“ entwickelte sich nicht wie erwartet, und es fanden sich keine Abnehmer für das Produkt. Daher war die Entwicklung des World Wide Web für das Team um James Gosling ein echter Glücksfall.

Ursprünglich war das Web entworfen worden, um den Austausch von wissenschaftlichen Informationen zwischen geographisch verteilten Forschergruppen zu erleichtern. Doch mit der Vorstellung des ersten Web-Browsers „NCSA Mosaic 1.0“ im April 1993 erreichte dieser Anwendungsbereich des Internets eine ungeahnte Bedeutung. Die übersichtliche Darstellung von Text und Grafik sowie die einfache Navigation zwischen verknüpften Informationen gaben nicht nur Wissenschaftlern, sondern einer viel breiteren Bevölkerungsschicht Zugang zu weltweit verteilten Daten.

Das Potential des World Wide Web wurde von Sun erkannt und das bisher erfolglose Oak im Sommer 1994 unter dem Namen „Liveoak“ neu ausgerichtet. Das World Wide Web schien das ideale Einsatzgebiet zu sein, da viele der ursprünglichen Designkriterien von Oak, wie Plattformunabhängigkeit, Zuverlässigkeit und Sicherheit, sich direkt auf diesen Internetdienst übertragen ließen. Im Herbst 1994 wurde ein erster Prototyp des Web-Browsers „WebRunner“ (später „HotJava“) fertiggestellt, der neben der Darstellung von Dokumenten auch die Übertragung und Ausführung von Oak-Programmen beherrschte. Es stellte sich allerdings heraus, daß der Name „Oak“ bereits urheberrechtlich geschützt war, da er von einer anderen Programmiersprache benutzt wurde. Die Sprache erhielt daher den neuen Namen „Java“, ein im amerikanischen Englisch umgangssprachlicher Ausdruck für Kaffee. Das Markenzeichen ermöglichte auch das Design des inzwischen bekannten Logos: die dampfende Kaffeetasse.

Das Java-System und der Web-Browser HotJava wurden im Mai 1995 auf der SunWorld '95 der Öffentlichkeit vorgestellt und stießen dort auf reges Interesse. Im Laufe des Jahres 1995 lizenzierten mehrere Hersteller, z.B. Netscape, Oracle, IBM und Microsoft, die Technologie und kündigten an, Java in ihre Produkte zu integrieren. Zum heutigen Zeitpunkt existiert für die meisten Betriebssysteme (z.B. diverse UNIX-Derivate, MS Windows 95/NT, IBM OS/2, ...) zumindest ein Web-Browser, der Java unterstützt.

## **4.2 Exkurs: World Wide Web**

Das World Wide Web, auch WWW, Web oder W3 genannt, ist ein Internetdienst, der es ermöglicht, elektronische, aber geographisch verteilte Dokumente miteinander zu verknüpfen. Die Entwicklung des Web begann 1989 am CERN (*Centre Européen de Recherches Nucléaires*), dem europäischen Zentrum für Kernphysik in Genf, aus der Notwendigkeit, Forschungsberichte zwischen weltweit arbeitenden Wissenschaftlern schneller und effizienter auszutauschen. Als Ergebnis dieser Entwicklung wurde im April 1993 die erste graphische Benutzeroberfläche „NCSA Mosaic 1.0“ für das Web der Öffentlichkeit vorgestellt. 1994 unterzeichneten CERN und MIT (*Massachusetts Institute of Technology*) eine Vereinbarung zur Gründung des World Wide Web Consortium, kurz W3C, einer Organisation zur Weiterentwicklung und Standardisierung des WWW.

Den Kern des World Wide Web bilden die beiden Standards HTML (*Hypertext Markup Language*) und HTTP (*Hypertext Transfer Protocol*). Die Dokumente des Web, auch Webseiten bzw. *webpages* genannt, bestehen aus Text und Formatierungsbefehlen (*Hypertext*) und basieren auf der Seitenbeschreibungssprache HTML. Die Sprache umfaßt Schriftauszeichnungen (z.B. fett, kursiv) sowie Mittel zur Textformatierung (z.B. zentriert) und Textgestaltung (z.B. Listen, Tabellen). Weiterentwickelte Versionen des HTML-Standards bieten die Möglichkeit der Einbettung von Bild-, Ton- und Videoinformationen.

Eine Besonderheit des Web stellen Verknüpfungen (*Hyperlinks* bzw. kurz *Links*) mit anderen Dokumenten dar. Ein solcher Link ist eine HTML-Anweisung innerhalb einer Webseite, der einen Verweis auf ein anderes Dokument enthält. Wird ein Link durch den Benutzer ausgewählt, so zeigt der Web-Browser das Ziel der Verknüpfung an. Links werden vom Browser i.allg. graphisch hervorgehoben.

Technisch gesehen ist das World Wide Web ein Client/Server-System. Der Web-Browser ist der Client auf der Seite des Anwenders. Er dient dem Empfang und der Darstellung von Dokumenten des WWW. Bei einer Anforderung des Benutzers lädt der Browser mittels des HTTP-Protokolls die gewünschte Seite vom Web-Server, interpretiert die Seitenbeschreibung und stellt sie formatiert auf dem Bildschirm des Anwenders dar. Der Web-Server ist der Dienst-erbringer des WWW. Er beherbergt eine Sammlung von zusammengehörigen Webseiten, die auch *website* genannt wird. Ein Serverprozeß bearbeitet die Anfragen der Web-Browser und überträgt die angeforderten Seiten.

## 4.3 Das Java-System

### 4.3.1 Die Bestandteile

Java wird häufig nur als eine weitere Programmiersprache angesehen. Dabei ist die Sprache nur ein Teil des Gesamtkonzeptes von Java, das folgende Bestandteile umfaßt:

- **die Programmiersprache** - Sie ist eine objektorientierte Sprache, deren Syntax stark der von C++ ähnelt.
- **das Maschinenmodell** - Java-Programme werden nicht für einen bestimmten, physikalisch existierenden Rechner übersetzt, sondern für die *Java Virtual Machine* (JVM). Diese stellt eine Beschreibung eines Rechners dar, auf dem übersetzte Java-Programme ablaufen können. Ein Java-Interpreter realisiert diese Maschine auf einem tatsächlich existierenden Rechner. Dieses Maschinenmodell der JVM ist ein zentrales Merkmal von Java, das verschiedene Eigenschaften von Java erst möglich macht.
- **die Entwicklungsumgebung** - Um Java-Programme erstellen, übersetzen und ablaufen lassen zu können, sind eine Reihe von Anwendungen notwendig, die zusammen eine

Java-Entwicklungsumgebung bilden. Minimal werden ein Java-Compiler, der im Quelltext vorliegenden Java-Programme in sogenannte `class`-Dateien übersetzt und ein Java-Interpreter, der die `class`-Datei ausführt, benötigt. Häufig kommt dabei das JDK (*Java Development Kit*) von Sun zum Einsatz, das für verschiedene Betriebssysteme kostenlos zur Verfügung steht.

#### 4.3.2 Die Eigenschaften

Die Eigenschaften von Java sind ausführlich in einem „White Paper“ von Sun [SUN95] in Form von elf Schlagworten beschrieben. Diese Schlagworte und die Kernpunkte werden hier in knapper Form wiedergegeben. Einige zentrale Begriffe, wie Java-Compiler, Java-Interpreter und Java Virtual Machine, werden zunächst informal verwendet und im nachfolgenden Abschnitt 4.3.3 näher erläutert. Die Schlagworte sind im einzelnen:

- **einfach:** Java ist ähnlich aufgebaut wie die Programmiersprachen C und C++, die heutzutage bei vielen Softwareprojekten eingesetzt werden und mit denen viele Programmierer bereits vertraut sind. Durch die Nutzung vorhandener Kenntnisse ist der Einstieg in Java relativ leicht zu bewältigen. Um das System so verständlich wie möglich zu machen, verzichtet Java auf selten benutzte Konstrukte von C++. Das trifft auch auf Elemente zu, die für Compilerbauer schwer zu implementieren oder für Programmierer schwer zu erlernen bzw. anzuwenden sind, wie z.B. das Überladen von Operatoren, Zeiger und Zeigerarithmetik. Weiterhin verfügt Java über eine automatische Speicherfreigabe (*garbage collection*), die den fehlerträchtigen Punkt der Speicherverwaltung für den Programmierer wesentlich vereinfacht.

- **objektorientiert:** Das objektorientierte Konzept von Java ist dem von C++ sehr ähnlich. Trotzdem gibt es entscheidende Unterschiede. C++ ist eine „hybride“ Sprache, bedingt durch die Abwärtskompatibilität zu C. Das gestattet einen Programmierstil, der die objektorientierten Eigenschaften von C++ nicht nutzt. Java dagegen ist eine „rein“ objektorientierte Sprache, die häufig mit dem Schlagwort „everything is an object“ gekennzeichnet wird. Sämtliche Elemente in Java, mit Ausnahme der primitiven Datentypen, liegen in Form eines Klassenobjektes vor und können so die Vorteile der Objektorientierung (z.B. Kapselung, Vererbung) nutzen.

- **verteilt:** Java verfügt über eine umfangreiche Bibliothek von Netzwerkroutrinen, die verbindungslose und verbindungsorientierte Kommunikation ermöglichen. Durch die Unterstützung verschiedener Protokolle der TCP/IP-Suite (z.B. HTTP, FTP, UDP, TCP) ist der Zugriff auf Daten über das Netz genauso einfach wie auf Dateien des lokalen Systems.

- **robust:** Java ist für Programme konzipiert, die in verschiedener Hinsicht zuverlässig sein müssen. Deshalb enthält Java ein mehrstufiges Konzept zur Vermeidung von Programmierfehlern, das viel Wert auf das frühe Feststellen von möglichen Problemen und fehlerträchtigen Situationen legt. Eine der häufigsten Fehlerquellen in C/C++-Programmen ist der fehlerhafte Umgang mit Zeigern. Sie wird durch das Sprachdesign von Java ausgeschlossen. Anstatt einer

Zeigerarithmetik bietet Java echte Klassenobjekte, z.B. in Form einer Array-Klasse oder Vector-Klasse. Durch eine strenge, statische Typprüfung findet der Compiler während der Übersetzung eine ganze Reihe weiterer möglicher Fehler, z.B. fehlerhafte bzw. unzulässige Zuweisungen. Schließlich finden Überprüfungen auch während des Programmablaufs durch das Laufzeitsystem der JVM statt.

- **sicher:** Java ist für den Einsatz in Netzwerkumgebungen gedacht. Daher wurde bereits beim Entwurf die Problematik Sicherheit berücksichtigt. Ein mehrstufiges Sicherheitssystem greift bei der Programmierung, der Programmübertragung und Programmabarbeitung in Java ein. Die einzelnen Mechanismen werden in Abschnitt 4.4 beschrieben.

- **architekturneutral:** Der Einsatz von Applikationen in heterogenen Netzwerken ist i.allg. problematisch, da die angeschlossenen Systeme auf unterschiedlichen Rechnerarchitekturen und Betriebssystemen basieren. Um Java-Programme trotzdem auf allen Plattformen einsetzen zu können, erzeugt der Compiler ein architekturneutrales Objektdateiformat, das aus sogenannten Bytecode-Instruktionen besteht. Damit kann der übersetzte Code auf allen Plattformen ausgeführt werden, für die ein Java-Laufzeitsystem zur Verfügung steht.

Der Bytecode ist vergleichbar mit herkömmlichen Maschinenbefehlen. Im Unterschied dazu ist er jedoch nicht für eine bestimmte Rechnerarchitektur gedacht, sondern für eine „Virtuelle Maschine“ (VM), die relativ leicht auf einem bestehenden System nachgebildet werden kann. Der Java-Interpreter stellt eine Implementierung der VM für eine bestimmte Plattform dar. Er interpretiert während der Ausführung des Java-Programms den Bytecode und übersetzt ihn in maschinenspezifische Befehle. Es ist somit nicht mehr notwendig, spezielle Varianten einer Anwendung für die unterschiedlichen Plattformen zu entwickeln.

Diesem Vorteil der Maschinenunabhängigkeit stehen jedoch deutliche Geschwindigkeitseinbußen bei der Interpretation von Bytecode, bezogen auf die Ausführung von nativem Code<sup>10</sup>, gegenüber. Dieser Nachteil kann durch sogenannte *Just-In-Time-Compiler* (JIT) zumindest teilweise gemindert werden. Der JIT-Compiler ist eine Erweiterung der Java-Laufzeitumgebung. Er übersetzt den Bytecode während der Ausführung in nativem Code und erreicht so fast die Arbeitsgeschwindigkeit herkömmlicher C++-Programme.

- **portabel:** Die Portabilität von Java basiert nicht nur auf der Anwendung der architekturneutralen JVM. Im Gegensatz zu herkömmlichen Sprachen, wie z.B. C, gibt es keine implementierungsabhängigen Erscheinungen. Die Größe sämtlicher primitiver Datentypen ist definiert. Somit ist der Wertebereich der Variablen auf jeder Plattform gleich.

Um eine sinnvolle Anwendungsprogrammierung zu ermöglichen, sind auch in Java systemabhängige Funktionen, z.B. für den Dateizugriff, notwendig. Diese Funktionen sind Teil der

---

<sup>10</sup> nativer Code - Programmcode, der auf einer bestimmten Rechnerarchitektur direkt durch die CPU ausgeführt werden kann

Java-Klassenbibliotheken und basieren auf portablen Schnittstellen. Damit können allerdings keine spezifischen Eigenschaften eines einzelnen Systems unterstützt werden. Die Klassenbibliotheken bieten somit nur den „kleinsten gemeinsamen Nenner“ aller Systeme.

Auch das Java-System von Sun selbst ist gut portierbar. Der Java-Compiler ist in Java geschrieben, der Java-Interpreter, d.h. das Laufzeitsystem, in ANSI-C.

- **interpretiert:** Der Java-Bytecode wird vom Java-Interpreter direkt interpretiert, d.h. jeder Befehl im Bytecode wird einzeln dekodiert und ausgeführt. Somit kann der Bytecode auf jeder Maschine ausgeführt werden, für die ein Java-Laufzeitsystem zur Verfügung steht.

Das Verfahren der Interpretation führt zu den bereits beschriebenen Geschwindigkeitseinbußen. Die Ursache liegt in den wiederholten Übersetzungen des Interpreters, wenn ein Programmabschnitt mehrfach ausgeführt wird. Besonders deutlich tritt diese Problematik bei der Abarbeitung von Schleifen auf. Interpreter haben jedoch auch Vorteile. Sie sind i.allg. leichter zu implementieren und bieten dem Programmierer während der Entwicklungsphase eine vereinfachte Fehlersuche.

- **schnell:** Die Geschwindigkeit des interpretierten Bytecodes reicht für den derzeit häufigsten Einsatz von Java, die Einbettung von kleinen Programmen in eine Webseite, aus. Trotzdem finden sich genügend Situationen, die eine höhere Geschwindigkeit erfordern. Die Leistung des Interpreters ist für umfangreiche Applikationen noch unzureichend.

Die Lösung liegt in der Anwendung von JIT-Compilern (siehe Abschnitt 4.3.4), die von IBM, Microsoft, Symantec und Borland angeboten werden. Sun entwickelt derzeit einen eigenen JIT unter dem Namen „HotSpot“, dessen Erscheinen für das dritte Quartal 1998 angekündigt ist. Die Geschwindigkeitssteigerungen beim Einsatz eines JIT sind im Vergleich zum Java-Interpreter beachtlich. Sie werden mit dem Faktor 10-20 angegeben, vereinzelt wird auch der Faktor 50 genannt. Da derzeit noch kein Standard-Benchmark für den Vergleich der unterschiedlichen JVMs existiert, kommt häufig die Software „CaffeineMark“ der Firma „Pendragon Software“<sup>11</sup> zum Einsatz.

- **multithreading:** Multithreading ist die Fähigkeit, Teile eines Programms parallel<sup>12</sup> ablaufen zu lassen. Die einzelnen Teile werden als Threads (*lightweight processes*) bezeichnet. Sie teilen sich im Gegensatz zu Prozessen ihre Datenbereiche und ermöglichen so einen einfachen Informationsaustausch. Multithreading sollte nicht mit Multiprocessing bzw. Multitasking verwechselt werden, bei dem verschiedene Programme (also Prozesse bzw. Tasks) parallel ablaufen. Die Threads werden u.a. vom Laufzeitsystem benutzt. Der beschriebene „garbage collector“ zur automatischen Speicherfreigabe ist ein Thread der JVM.

---

<sup>11</sup> verfügbar unter: <http://www.webfayre.com/pendragon/cm3/index.html>

<sup>12</sup> Echte Parallelität ist natürlich nur auf einem Multiprozessorsystem möglich, ansonsten handelt es sich um eine Quasi-Parallelität.

Während die Programmierung von parallelen Programmteilen in herkömmlichen Sprachen schwierig und fehlerträchtig ist, stellt Java Threads als Sprachbestandteil bereit, d.h. jede Java-Implementierung verfügt über Threads. Dies wird sowohl durch entsprechende sprachliche Mittel, als auch durch spezielle Anweisungen zur Synchronisation im Befehlssatz der JVM gewährleistet.

- **dynamisch:** Java verwendet das Konzept des „late binding“, d.h. die externen Referenzen von Instanzvariablen und Methoden werden erst zur Laufzeit aufgelöst. Dieses Verfahren vermeidet einen gesonderten Link-Vorgang während der Programmerstellung und bietet eine größere Flexibilität gegenüber Sprachen wie C und C++. Da die referenzierten Objekte zur Laufzeit über ihren Namen und nicht über einen Offset (relative Positionsnummer) angesprochen werden, ist es ohne Neukompilierung möglich, externe Klassen auszutauschen bzw. zu ändern.

Die Eigenschaften des Java-Systems scheinen auf den ersten Blick sehr innovativ zu sein. Bei genauerer Betrachtung stellt sich jedoch heraus, daß die verschiedenen Ideen und Konzepte von Java keineswegs neu sind. Sie stellen nur eine Wiederbelebung bereits bekannter Ansätze, die zum Teil bis in die 70er Jahre zurückreichen, dar.

So sind Portabilität, Robustheit gegenüber Programmfehlern und statische Typsicherheit seit Jahren Merkmale bei der Entwicklung höherer Programmiersprachen. Hier schneidet Java nur im direkten Vergleich zu C oder C++ besser ab. Auch die Idee, vom Compiler keinen nativen Code, sondern maschinenunabhängige Instruktionen erzeugen zu lassen, die auf einer virtuellen Maschine ausgeführt werden, ist nicht neu. Das Verfahren wurde bereits Mitte der 70er Jahre in UCSD-p-Systemen (*University of California, San Diego*) mit UCSD-Pascal und UCSD-Basic angewendet und fand später auch in Smalltalk-Systemen Verwendung. Andere Eigenschaften, wie objektorientiert, automatische Speicherverwaltung, dynamisches Linken sowie die Kombination von Programmiersprache und Laufzeitumgebung sind zum Beispiel dem Oberon-System von Niklaus Wirth sehr ähnlich.

Das Java-System ist also eine Kombination verschiedener bekannter Konzepte. Es ist jedoch keineswegs so „revolutionär“, wie es Sun und einige Autoren in ihren Publikationen darstellen.

#### **4.3.3 Compiler, Interpreter und Virtuelle Maschine**

Die bereits im letzten Abschnitt informal verwendeten Begriffe Java-Compiler, Java-Interpreter und Java Virtual Machine sollen hier genauer beschrieben werden. Um die Begriffe zu verdeutlichen, wird eine Analogie zur herkömmlichen Programmentwicklung gezogen. Als Beispiel sei die Programmierung mit den Sprachen C und Lisp angenommen, wobei sich die Ausführungen auch auf andere Sprachen übertragen lassen.

Bei der Programmentwicklung erstellt der Programmierer zunächst den Quelltext seines C-Programmes. Dazu kann er im einfachsten Fall einen beliebigen Texteditor verwenden. Anschließend compiliert er den Quelltext mit einem C-Compiler. Dieser liest den Programmtext ein und übersetzt ihn in Maschinenbefehle des jeweiligen Zielsystems. Die generierten Anweisungen sind dabei spezifisch für einen bestimmten Prozessortyp, z.B. für die Intel Pentium-CPU. Einige Elemente des compilierten Programms, wie Funktionsaufrufe, Speichermodell und Dateiformat, sind zudem abhängig vom verwendeten Betriebssystem. Bei der Ausführung des Programms kann die CPU die generierten Instruktionen direkt abarbeiten. Es sind somit keine Zwischenschritte, z.B. für eine Konvertierung, notwendig. Diese Verarbeitung führt zu hohen Ablaufgeschwindigkeiten.

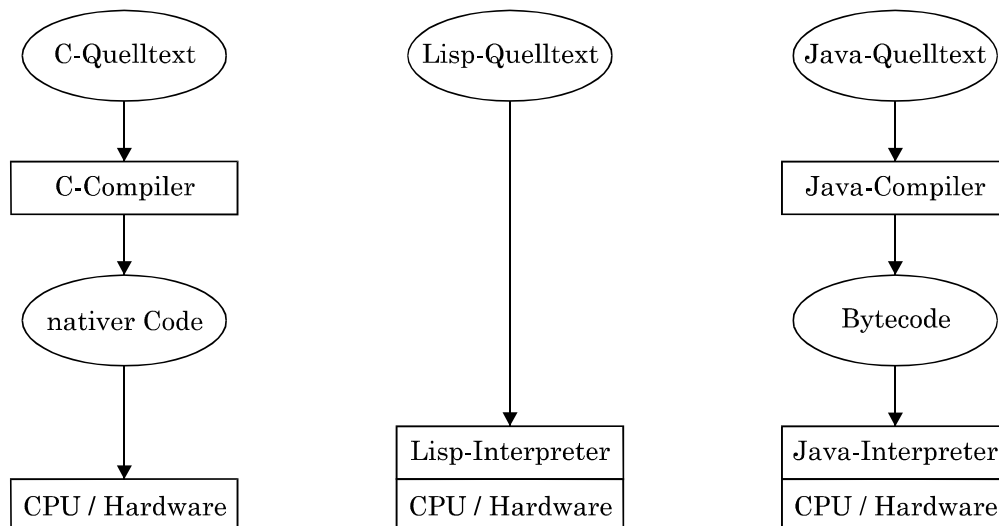
Neben den compilierten Sprachen gibt es auch interpretierte Sprachen, zu denen u.a. Lisp gehört. Lisp-Programme werden vor der Programmausführung nicht von einem Compiler in Maschinenbefehle übersetzt. Statt dessen werden die Anweisungen zur Programmlaufzeit durch einen Interpreter einzeln bearbeitet, syntaktisch überprüft und in Maschinencodes übertragen. Interpretierte Sprachen bieten den Vorteil, daß Änderungen am Programmtext schneller ausgetestet werden können, da ein zusätzlicher Compilerlauf zwischen Programmänderung und -ausführung entfällt. Ihr Nachteil ist die meist sehr aufwendige Übersetzung der jeweiligen Programmiersprache in Maschinenbefehle, die zur Laufzeit des Programmes stattfindet und den Ablauf verlangsamt. Häufig findet jedoch auch bei Interpreter-sprachen vor der eigentlichen Programmausführung eine Vorverarbeitung, d.h. eine Art Compilierung, der Quelltexte statt. Dabei werden die Anweisungen syntaktisch überprüft und in einen Zwischencode, auch p-Code genannt, übersetzt (Stichwort: Tokenisierung). Dieser kann zur Laufzeit vom Interpreter mit deutlich geringerem Aufwand als bei reinen interpretierten Sprachen in Maschinenbefehle übersetzt werden.

Dieses Modell kann auch auf Java übertragen werden, denn wenn der Zwischencode hinreichend allgemein und maschinenunabhängig ist, kann er von Interpretern auf verschiedenen Maschinen zur Ausführung gebracht werden. Der Java-Compiler übersetzt die in Java programmierte Quelltextdatei ebenfalls in einen Zwischencode, den sogenannten Bytecode. Dabei ist die Analogie zu compilierten Sprachen stärker als zu den vorverarbeiteten, interpretierten Sprachen, denn in Java ist nicht nur der Bytecode, sondern auch ein zugehöriges Maschinenmodell spezifiziert. Der Java-Compiler generiert mit dem Bytecode Maschinenanweisungen für einen bestimmten Prozessortyp, die Java Virtual Machine. Dabei handelt es sich nicht um einen real existierenden Prozessor, sondern lediglich um eine Spezifikation. Dieses Modell kann jedoch auf einem existierenden Rechner mittels Software nachgebildet werden. Dies ist Aufgabe des Java-Interpreters. Er stellt eine Implementierung der Spezifikation der JVM dar. Somit können auf jedem Rechner und zugehörigem Betriebssystem, für die eine solche JVM-Implementierung existiert, Java-Programme ausgeführt werden.



Auch eine Umsetzung der Spezifikation in Hardware ist möglich. Sun arbeitet seit 1996 an einem Java-Prozessor namens „picoJava“, der in NCs (*Network Computer*), PDAs (*Personal Digital Assistant*), Mobiltelefonen und Pagern eingesetzt werden soll. Mit ersten Mustern wird 1998<sup>13</sup> gerechnet.

Der beschriebene Zusammenhang von Java mit compilierten bzw. interpretierten Sprachen ist noch einmal in Abbildung 14 zu sehen. Die Abbildung zeigt die einzelnen Stufen zwischen der Übersetzung und der Abarbeitung des Programms.



**Abbildung 14: Vergleich von Java mit compilierten und interpretierten Sprachen**

#### 4.3.4 Just-In-Time-Compiler (JIT)

Java-Programme basieren auf einem architekturneutralen Bytecode, der auf verschiedenen Plattformen innerhalb einer JVM ausgeführt werden kann. Dazu muß der Bytecode allerdings in native Prozessoranweisungen auf der Zielmaschine übersetzt werden, da die derzeit vorrangig eingesetzten CPUs, z.B. Intel x86, IBM-Motorola PowerPC, Sun SPARC u.a., die Bytecode-Instruktionen nicht verarbeiten können. Diese Übersetzungen kosten Zeit und verringern die Ablaufgeschwindigkeit von Java-Programmen zum Teil erheblich.

Die Ursache für die mangelhafte Leistung ist die Implementierung der JVM in Form eines Java-Interpreters. Dieser übersetzt die Bytecode-Instruktionen nacheinander während der Laufzeit des Programmes und muß diesen Schritt mehrfach wiederholen, wenn eine Anweisung mehrfach ausgeführt werden soll. Dieses Problem tritt besonders kraß bei der Abarbeitung von Schleifen auf. Somit reicht die Leistung des Java-Interpreters für umfangreiche bzw. rechenintensive Anwendungen nicht aus. Trotzdem basieren verschiedene JVMs, u.a. die von Sun, auf diesem Prinzip, da Interpreter relativ leicht zu implementieren und portieren sind.

---

<sup>13</sup> Angabe in der Zeitschrift c't, 12/97, Seite 34

Einen effektiven Weg, um die Geschwindigkeit von Java-Programmen zu erhöhen, bietet der Einsatz eines Just-In-Time-Compilers, kurz JIT genannt. Der JIT stellt eine spezielle Form der Virtuellen Maschine dar, welche den Bytecode unmittelbar vor der Ausführung in native Prozessoranweisungen der Zielmaschine übersetzt. Dabei liegen die generierten Anweisungen lediglich im Speicher vor, so daß der Übersetzungsvorgang beim nächsten Start des Java-Programmes wiederholt werden muß. Im Ergebnis bleibt ein portabler Bytecode, bei dem insbesondere die Abarbeitung von Schleifen beschleunigt wird. Die typischen Geschwindigkeitssteigerungen werden im Vergleich zu einem Java-Interpreter mit dem Faktor 10 bis 20, vereinzelt sogar mit dem Faktor 50 angegeben.

JIT-Compiler haben allerdings auch mit einigen Problemen zu kämpfen. So muß die Compilierung im Vergleich zu herkömmlichen Compilern sehr schnell erfolgen, damit sich der Start des Java-Programmes nur unmerklich verzögert. Da bleibt nur wenig Spielraum für sonst übliche Optimierungen, z.B. die Benutzung von Registervariablen, die Berechnung konstanter Ausdrücke und die Entfernung unbenutzter Variablen. Mitunter kann die Compilierung und Optimierung einzelner Codestücke länger dauern als ihre spätere Abarbeitung.

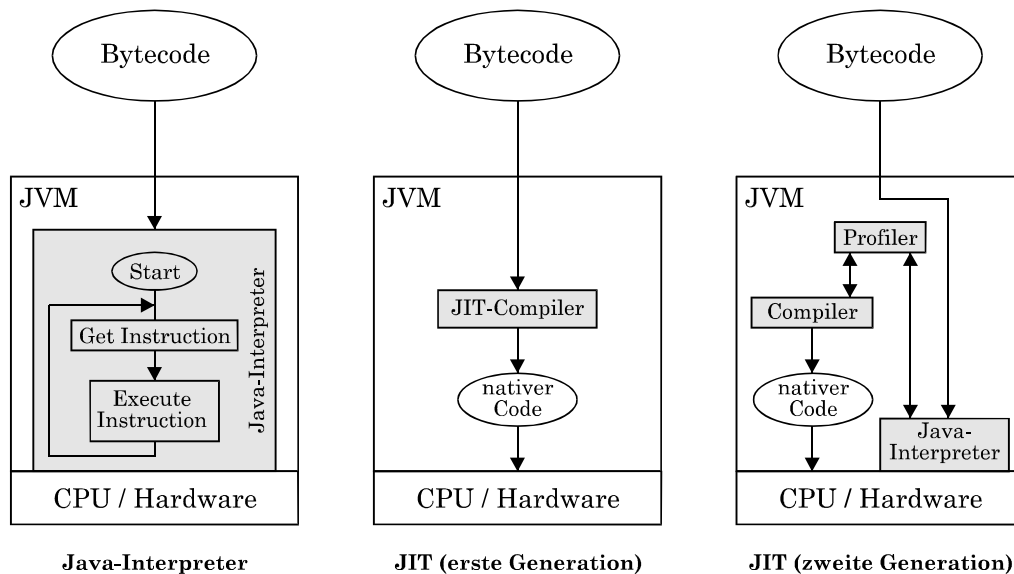
Derzeit unterscheidet man zwei Arten von JITs:

- die „klassischen“ JITs der ersten Generation und
- die dynamischen JITs der zweiten Generation.

Zur ersten Generation der JIT-Compiler gehören u.a. die auf dem Markt verfügbaren JVMs von Symantec, Borland und Microsoft. Sie stellen den „klassischen“ JIT dar, der den Bytecode vor der Ausführung vollständig übersetzt. An einem dynamischen JIT-Compiler der zweiten Generation wird derzeit bei Sun unter dem Namen „HotSpot“ gearbeitet. Er stellt eine Kombination von Java-Interpreter, JIT-Compiler und Profiler dar. Beim Start des Java-Programmes führt der Interpreter den geladenen Bytecode aus. Dies vermeidet die Verzögerung gegenüber „klassischen“ JITs. Während der Ausführung überwacht der Profiler die Abarbeitungszeit aller Methoden. Stellt er dabei fest, daß eine der Methoden besonders lange braucht, so weist der Profiler den JIT-Compiler an, diese Methode zu übersetzen und zu optimieren. Bei jedem weiteren Aufruf der Methode werden anschließend die generierten, nativen Prozessoranweisungen ausgeführt. Dieses Verfahren ist auch unter dem Stichwort „adaptive Optimierung“ bekannt. Es ist effizienter als die Arbeit der herkömmlichen JITs, da nur die kritischen Abschnitte (auch „*hot spots*“ genannt) der Applikation übersetzt und optimiert werden. Nutzlose Optimierungen für Programmabschnitte, die nur einmal durchlaufen werden, können dadurch vermieden werden. Sun hofft mit der JVM „HotSpot“ und Java in die Leistungsbereiche herkömmlicher C++-Programme vorstoßen zu können.

Die Abbildung 15 zeigt den Aufbau der drei beschriebenen JVMs, der Java-Interpreter sowie ein JIT der ersten und zweiten Generation. Sie stellt im linken Bildteil schematisch die Ar-

beitsweise des Interpreters dar und listet in den beiden anderen Bildteilen die Bestandteile der JITs auf.



**Abbildung 15: Aufbau der unterschiedlichen JVMs**

### 4.3.5 Applikationen und Applets

In den bisherigen Abschnitten wurde nur allgemein von Anwendungen in Java gesprochen. Dabei unterscheidet man jedoch zwischen zwei unterschiedlichen Formen von Programmen:

- Applikationen und
- Applets.

Eine Java-Applikation ist ein in Java-Bytecode übersetztes Programm, das mit Anwendungen, die in einer konventionellen Programmiersprache erstellt wurden, vergleichbar ist. Eine Applikation wird auf dem lokalen Rechner ausgeführt und unterliegt, im Gegensatz zu Applets, keinen Ausführungsbeschränkungen.

Typischerweise wird eine Java-Applikation von der Kommandozeile gestartet. Je nach Betriebssystem kann das Programm entweder direkt ausgeführt werden oder es muß der Programmname der Java-Umgebung (z.B. ein Java-Interpreter) angegeben werden, die als Parameter den Namen der auszuführenden Java-Klasse erhält. Im ersten Fall erkennt das Betriebssystem, daß eine Java-Applikation ausgeführt werden soll (z.B. an der Dateierweiterung `.class`) und startet die notwendige JVM selbst.

Jede Java-Applikation muß eine Methode `main()` implementieren, die den Start- und Eintrittspunkt für das Programm bildet. Die Abbildung 16 zeigt eine einfache Applikation, die den String „Hello World!“ in die Standardausgabe druckt.

```

public class HelloWorldApp extends Object {
    public static void main(String[] args) {
        System.out.println("Hello World !");
    }
}

```

#### **Abbildung 16: Beispiel einer Java-Applikation**

Ein Applet ist eine besondere Form einer Applikation. Es sind kleine Programme, die im compilierten Java-Bytecode vorliegen und in HTML-Dokumente eingebettet werden. Das statische Dokument wird durch die Einbettung einer ausführbaren Komponente um interaktive Elemente erweitert.

Der Benutzer kann mittels eines Web-Browsers ein HTML-Dokument von einem entfernten Web-Server über ein Netzwerk abrufen und darstellen. Ein Java-fähiger Web-Browser überträgt bei einer solchen Anfrage auch die eingebetteten Applets über das Netz und führt sie beim Empfänger lokal aus. Dazu implementiert der Browser eine JVM, die den Ausführungsrahmen für das Applet bildet. Im Normalfall merkt der Anwender nicht, daß ein solchen Programm geladen wird. Er muß sich somit um keine Details, wie Protokolle, Pfadnamen o.ä. kümmern. Diese Vorgehensweise ist für den Benutzer sehr bequem. Sie wirft allerdings die Frage nach der Sicherheit solcher ausführbarer Dokumente auf, ein Problem, das im nachfolgenden Abschnitt 4.4 näher betrachtet wird. Da das Ausführen von fremden und unbekannten Programmcode immer ein gewisses Sicherheitsrisiko darstellt, ist die Funktionalität von Applets beschränkt. So können Applets, im Gegensatz zu Applikationen, bestimmte Operationen, z.B. Zugriffe auf das Dateisystem, überhaupt nicht oder zumindest nur stark eingeschränkt ausführen.

Die Abbildung 17 zeigt ein einfaches Applet, das den String „Hello World !“ im zur Verfügung stehenden Ausgabebereich des Web-Browsers ausgibt.

```

import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
    public void init() {}
    public void start() {}
    public void stop() {}
    public void destroy() {}

    public void paint(Graphics g) {
        g.drawString("Hello World !", 50, 50);
    }
}

```

#### **Abbildung 17: Beispiel eines Java-Applets**

Den Start- und Eintrittspunkt bilden die Methoden `init()` und `start()`, die vom Browser aufgerufen werden. Eine Methode `main()` existiert im allgemeinen nicht. Dagegen verfügen Applets über eine Methode `paint()`, die vom Browser aufgerufen wird, wenn der dem Applet

zur Verfügung stehende Ausgabebereich aktualisiert werden soll. Dafür stehen die Methoden des AWT (*Abstract Window Toolkit*), der Java-Klassenbibliothek zur Programmierung von Fensteroberflächen, zur Verfügung. Den Abschluß bilden die beiden Methoden `stop()` und `destroy()`, die bei Beendigung des Applets ausgeführt werden.

Beide Varianten von Java-Programmen müssen sich nicht gegenseitig ausschließen. So können auch Applets eine Methode `main()` haben und Applikationen das AWT benutzen, auch wenn letzteres mit einigen Vorbereitungsarbeiten verbunden ist. So ist eine Kombination von Applikation und Applet möglich, die sowohl im Web-Browser als auch von der Kommandozeile ausgeführt werden kann.

#### **4.4 Das Sicherheitssystem von Java**

Einer der interessantesten Aspekte des Java-Systems ist die Fähigkeit, auf unterschiedlichen Plattformen Programmcode in Form von Applets von einem entfernten Web-Server zu laden und in einem Java-fähigen Web-Browser auszuführen. Dabei muß allerdings sichergestellt werden, daß der importierte Code die Integrität und Sicherheit des lokalen Rechners nicht verletzt, denn die potentiellen Gefahren sind umfangreich und sollten nicht unterschätzt werden. Sie reichen vom Ausspähen von Informationen bis hin zu umfangreichen Datenmanipulation. Dabei muß der geladene Code noch nicht einmal schädigende Absichten haben. Bereits Programmfehler können den Rechner beeinflussen, so daß Absturz oder Datenverlust die Folge sind.

Ein mehrstufiges Sicherheitssystem soll Ereignisse dieser Art in Java verhindern. Damit betreibt Java im Vergleich zu anderen Systemen, z.B. ActiveX von Microsoft, einen recht hohen Aufwand, um die Sicherheit des lokalen Systems des Anwenders zu gewährleisten. Grundsätzlich unterscheidet Java bezüglich der Sicherheit zwischen lokalen Programmen und Applets, die über das Netz geladen werden. Letztere laufen nur innerhalb einer hermetisch abgeschlossenen Umgebung - der Sandbox - ab, die sicherstellen soll, daß das Programm keinen Schaden anrichtet. Deshalb unterliegen Applets rigorosen Einschränkungen: Sie dürfen nicht auf das lokale Dateisystem zugreifen, keine fremden Programme starten und Netzwerkverbindungen nur zu dem Rechner aufnehmen, von dem sie geladen wurden. Um die Einhaltung dieser Spielregeln sicherzustellen, gibt es in Java folgende Sicherheitsinstanzen:

- das Sprachdesign,
- den Bytecode-Verifier,
- den Klassenlader und
- den Security-Manager.

Sie werden in den nachfolgenden Abschnitten näher erläutert.

#### 4.4.1 Das Sprachdesign

Bei der Definition der Sprache Java wurde besonderer Wert darauf gelegt, sprachbezogene Sicherheitslücken, die etwa von C und C++ her bekannt sind, zu schließen. Es darf nicht möglich sein, daß die Ausführung eines Programms den Rechner in einen undefinierten Zustand bringt. Ein fehlerhaftes Programm muß entweder während der Compilierung oder bei der Ausführung eine exakt definierte Fehlermeldung erzeugen. Um dies zu gewährleisten, ist das Problem der Sicherheit bereits beim Design der Sprache berücksichtigt worden.

Java ist ein objektorientiertes System, bei dem sämtliche Elemente, mit Ausnahme der primitiven Datentypen, in Form von Klassen realisiert werden. Das gestattet die Integration von Sicherheitschecks, die am Beispiel der Array-Klasse gezeigt werden sollen. Im Gegensatz zu manch anderer Sprache ist der Zugriff auf die Elemente eines Arrays nur über die Methoden der Klasse möglich. Dabei werden automatisch die Gültigkeit des Indexes geprüft und Zugriffe außerhalb der Bereichsgrenzen mit einer Exception abgewiesen. Weiterhin sind sicherheitsrelevante Klassen und Methoden der Java-API (*Application Programming Interface*) als „final“ deklariert, d.h. sie lassen sich nicht vererben und durch unsicheren Programmcode ersetzen.

Java realisiert den Zugriff auf Objekte über Referenzen. Das Fehlen von Zeigern und einer Zeigerarithmetik verhindert die unkontrollierte Adressierung von Speicherzellen. Zusätzlich können keine ungeprüften Typumwandlungen (*type casts*) durchgeführt werden. Die Konvertierung ist nur zwischen Objekten mit kompatiblen Typen möglich. Der Check findet sowohl statisch während der Übersetzung durch den Compiler als auch dynamisch zur Laufzeit statt.

Ein „vertrauenswürdiger“ Java-Compiler garantiert die Einhaltung dieser Sicherheitsrichtlinien und schützt so vor den Auswirkungen unbeabsichtigter Programmierfehler.

#### 4.4.2 Der Bytecode-Verifier

Die Richtlinien allein reichen jedoch für ein funktionierendes Sicherheitssystem noch nicht aus. Sie könnten durch fehlerhafte Compiler oder Manipulationen am Bytecode unterlaufen werden. Es ist auch durchaus möglich, einen Java-Compiler zu schreiben, der gezielt gegen die Richtlinien verstößt und so Code generiert, der in schädigender Absicht die Sicherheit umgeht.

Daher sind zusätzliche Checks von seiten des Browsers absolut notwendig, um die Sicherheit der Ausführungsumgebung zu gewährleisten. Der Bytecode wird solange als potentiell gefährlich angesehen, bis die Prüfungen das Gegenteil beweisen. Die Instanz, die diesen Beweis zu erbringen hat, ist der sogenannte Bytecode-Verifier<sup>14</sup>. Jede aus dem Netz stammende Java-Klasse durchläuft ihn, bevor sie in die JVM geladen wird.

---

<sup>14</sup> Zu deutsch etwa: Überprüfer, Kontrolleur. Der Begriff hat sich ähnlich wie Compiler und Interpreter in der Fachliteratur eingebürgert.

Dabei führt der Bytecode-Verifier folgende Tests durch:

1. Das Format der Klassendatei wird bezüglich Integrität und Struktur überprüft. Die Versionsnummer, die sich im Header der Datei befindet, muß kompatibel zu den anderen Java-Klassen und der JVM sein (Abwärtskompatibilität).
2. Die Einhaltung der Java-Spezifikation wird überprüft. Jede Klasse muß, mit Ausnahme der generischen Klasse `Object`, von einer Oberklasse abgeleitet sein. Als „final“ deklarierte Klassen und Methoden dürfen nicht vererbt oder überschrieben werden.
3. Der Bytecode wird einer detaillierten Untersuchung unterzogen. Dabei wird nach illegalen Bytecodes, unzulässigen Zuweisungen oder Typumwandlungen, Verletzungen der Zugriffsrechte (`protected`, `private`) und Operationen auf inkompatiblen Typen gesucht. Weiterhin werden die Parameter bei Methodenaufrufen geprüft und der Stack auf Über- und Unterläufe getestet.

Die Überprüfung des Bytecodes durch den Verifier ist eines der stärksten Sicherheitsmerkmale von Java, da der schadhafte Code hinreichend komplex sein müßte, um sämtliche Tests vor seiner eigentlichen Ausführung zu bestehen.

#### **4.4.3 Der Klassenlader**

Die Klassendateien, die den Verifier erfolgreich passiert haben, treffen anschließend auf die nächste Instanz des Sicherheitssystems von Java, den Klassenlader. Er ist ein Objekt der Klasse `java.lang.ClassLoader` und lädt angeforderte Java-Klassen in die JVM. Der Klassenlader behandelt ein Sicherheitsproblem, das in Zusammenhang mit dem „late binding“ auftritt. Mit dieser Eigenschaft wäre es möglich, geprüfte und vertrauenswürdige Klassen der Java-API im lokalen System auszublenden und durch manipulierte Klassen gleichen Namens von einem entfernten Server zu ersetzen.

Um dieses zu verhindern, benutzt der Klassenlader das Konzept der getrennten Namensräume (*name spaces*). Ein Namensraum steht für eine Menge von Java-Klassen mit gemeinsamer Herkunft. So existiert ein Namensraum für die Klassen des lokalen Systems<sup>15</sup> und jeweils ein Namensraum für Klassen von einer bestimmten Netzwerkadresse. Klassen mit gleichem Namen, aber unterschiedlicher Herkunft werden so in getrennte Namensräume eingefügt.

Wenn eine Klasse benötigt wird, sucht der Klassenlader immer zuerst im lokalen Namensraum und erst danach in den weiteren Namensräumen. Dadurch haben lokal installierte Klassen immer den Vorrang gegenüber Klassen, die vom Netzwerk geladen werden. Ein Ausblenden von sicherheitsrelevanten Klassen der Java-API ist nicht mehr möglich.

---

<sup>15</sup> Eine Java-Klasse gilt als lokal, wenn sie im Dateisystem des Rechners vom Anwender in einem durch die Umgebungsvariable `CLASSPATH` bezeichneten Directory liegt.

#### 4.4.4 Der Security-Manager

Der Security-Manager ist die letzte Instanz des Sicherheitssystems von Java. Er überwacht systemkritische Operationen auf dem lokalen Rechner und verhindert so z.B. die Zugriffe eines Applets auf das Dateisystem.

Der Security-Manager ist ein Objekt der Klasse `java.lang.SecurityManager`, der je nach Java-Implementierung unterschiedlich ausfallen kann. Er wird beim Start der JVM initialisiert und kann anschließend weder ersetzt noch verändert werden. Der Security-Manager legt die „Spielregeln“ für Java-Programme fest, wobei Web-Browser i.allg. sehr restriktiv eingestellt sind. So verhindert der Netscape Navigator folgende Operationen bei Applets:

- die Installation eines neuen Security-Managers,
- die Installation eines neuen Klassenladers,
- lesende oder schreibende Zugriffe auf das lokale Dateisystem,
- das Löschen von Dateien,
- das Ausführen externer Programme,
- die Einbindung nativer Bibliotheken sowie
- unbeschränkte Netzwerkkommunikation, d.h. Netzwerkverbindungen können nur zu dem Rechner aufgebaut werden, von dem das Applet geladen wurde.

Andere Security-Manager erlauben etwas feinere Einstellungen der Zugriffsrechte, z.B. durch Zugriffslisten (*Access Control Lists*), die vom Anwender konfiguriert werden können.

#### 4.4.5 Die Kosten der Sicherheit

Die Kosten der Sicherheitsmaßnahmen gehen größtenteils zu Lasten der Funktionalität von Applets. Zwar kosten die Tests auch Zeit, aber die meisten Prüfungen werden bereits beim Laden der Klasse durchgeführt und sind somit einmalige Aufwendungen. Der Zeitaufwand für die Tests während der Laufzeit kann i.allg. vernachlässigt werden.

Dagegen sind die funktionellen Einschränkungen schon beachtlich. Vielfach lassen sich bei umfangreichen Anwendungen mit Applets nur visuelle Aufgaben erledigen. Die eigentliche Programmlogik muß aufwendig in einem Client/Server-Modell realisiert werden. So ist es verständlich, daß in Entwicklerkreisen der Wunsch nach geringeren oder konfigurierbaren Sicherheitsrestriktionen laut wird.

Trotzdem, die Sicherheit des Java-Systems dient in erster Linie dem Anwender. Er soll vor fragwürdigen Programmcode aus unbekannten Quellen geschützt werden. So sollte auch langfristig das Thema „Sicherheit“ bei Java-Applets im Vordergrund stehen und nicht zugunsten schneller Lösungen aufgegeben werden.



## **5 Die Anwendung „JReport“**

In diesem Kapitel wird am Beispiel der Anwendung „Java Device Report“, nachfolgend nur noch kurz „JReport“ genannt, der Entwurf eines architekturneutralen Reportgenerators in Java beschrieben und seine Implementierung erläutert. Er ermöglicht die Dokumentation von Parametern und Einstellungen von Netzwerkkomponenten auf der Grundlage des Managementprotokolls SNMP. Dabei werden die einzelnen Bestandteile der Anwendung erklärt und Schwierigkeiten bei der Umsetzung der plattformunabhängigen Lösung aufgezeigt.

### **5.1 Das Konzept der Anwendung**

#### **5.1.1 Vorbemerkungen**

Sämtliche nachfolgenden Aussagen zu Java bzw. zum Java-System beziehen sich auf das JDK von Sun in der Version 1.0.2, das seit dem September 1996 als vorläufiger Java-Standard festgeschrieben ist. Diese Java-Version wird allgemein unterstützt und bildet die Grundlage der meisten VMs in den Java-fähigen Web-Browsern.

Neue und erweiterte Java-Versionen (1.1.x) werden aufgrund der noch recht häufigen Änderungen erst langsam in die bestehenden Browser integriert. Vorreiter ist dabei Sun mit dem Web-Browser „HotJava“. Die etablierten Vertreter, Netscape Navigator und Internet Explorer, bieten dagegen keine oder nur mangelhafte Java-1.1-Unterstützung.

Die Java-Version 1.1.x ist prinzipiell abwärtskompatibel zu Java 1.0.2, so daß alte Java-Anwendungen auch unter der neuen Umgebung ausgeführt werden können. Aufgrund von Veränderungen der Java-API in der Version 1.1.x warnt der Compiler bei der Übersetzung von Quelltexten jedoch vor potentiell veralteten und in späteren Java-Version nicht mehr unterstützten Methoden.

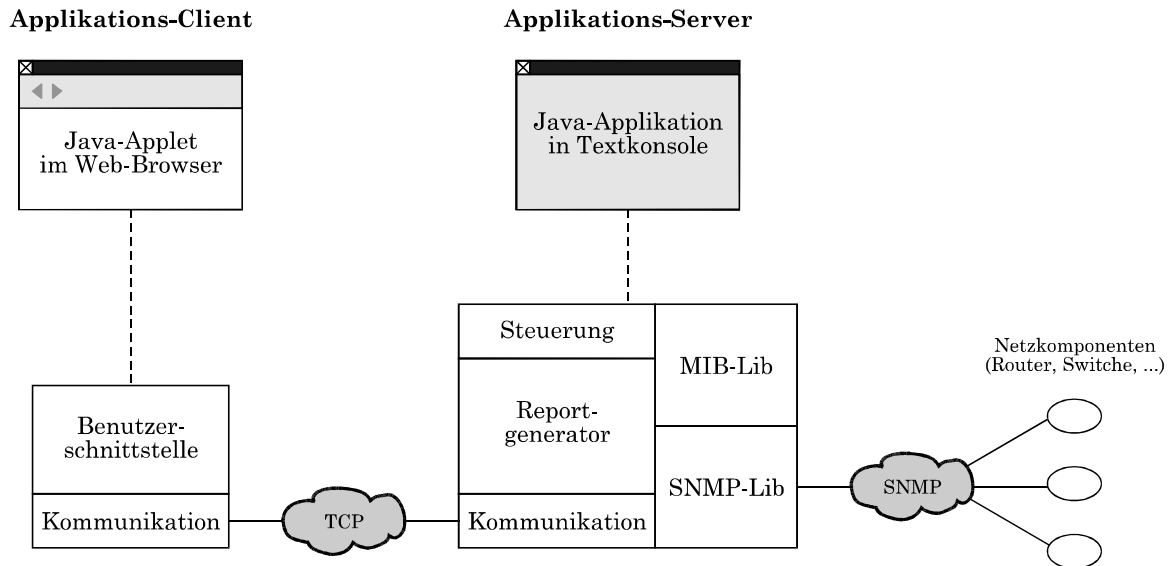
#### **5.1.2 Die Entwicklungsumgebung**

Die Anwendung JReport wurde mit dem Java Development Kit (JDK) 1.0.2 von Sun unter Microsoft Windows NT 4.0 erstellt. Da diese Software jedoch nur einfache Entwicklungswerkzeuge in Form von kommandozeilenbasierten Anwendungen zur Verfügung stellt, wurde für die Entwicklung der grafischen Benutzerschnittstelle das Produkt „Visual Café 1.0“ von Symantec eingesetzt. Alle weiteren Arbeiten fanden unter der Entwicklungsumgebung „Kawa“ statt, die unterschiedliche JDKs einbinden kann.

Für den Test der Anwendung während der Entwicklungsphase wurde vorrangig der Netscape Navigator verwendet. Details zu weiteren Testumgebungen finden sich im Abschnitt 5.2.4.

### 5.1.3 Die Struktur der Anwendung

Die Anwendung JReport ist im wesentlichen aus zwei Komponenten aufgebaut, dem Applikations-Client und dem Applikations-Server. Beide zusammen realisieren ein Client-/Server-Modell, dessen Struktur in der Abbildung 18 dargestellt ist.



**Abbildung 18: Struktur der Anwendung JReport**

Der Applikations-Client läuft innerhalb eines Web-Browsers und besteht im wesentlichen aus einem Java-Applet. Der Client realisiert die grafische Benutzerschnittstelle der Anwendung. Er nimmt die Eingaben des Benutzers entgegen und schickt sie zur Verarbeitung an den Applikations-Server. Dessen Ergebnisse, d.h. die generierten Berichte, werden über den Client im Web-Browser dargestellt.

Der Applikations-Server ist der Dienstbringer der Anwendung. Er implementiert den eigentlichen Reportgenerator und übernimmt die SNMP-Managementkommunikation mit den gewünschten Netzkomponenten. Der Server ist MultiClient-fähig und kann somit seine Dienste mehreren Klienten gleichzeitig zur Verfügung stellen. Er basiert auf einer Java-Applikation, die kommandozeilenbasiert in einer Textkonsole abläuft. Die wesentlichen Komponenten des Servers sind, neben dem bereits genannten Reportgenerator, zwei Libs (Softwarebibliotheken), welche die SNMP- und MIB-Schnittstellen implementierten. Die einzelnen Bestandteile werden in den nachfolgenden Abschnitten näher erläutert.

Die Verwendung des Client-/Server-Modells für die Anwendung JReport wird primär durch zwei Punkte verursacht:

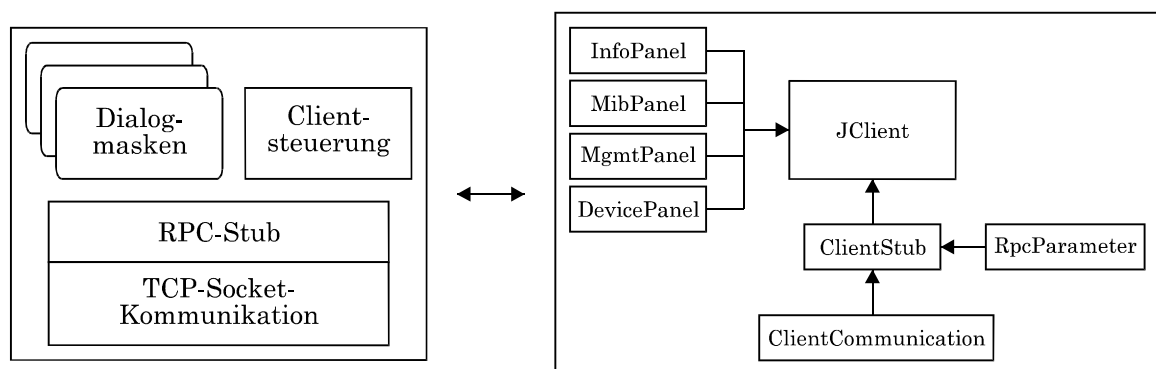
1. die Forderung nach dem Dokumentenformat HTML für Berichte und
2. die Sicherheitsrestriktionen von Java.

Der Einsatz von HTML für die generierten Berichte erfordert die Benutzung eines Web-Browsers für die Darstellung der Dokumente, da das Java-API für diesen Zweck bisher keine geeigneten Methoden bereitstellt. Um einen geschlossenen Anwendungsrahmen zu erreichen, bietet sich der Einsatz von Java-Applets innerhalb des Browsers an. Diese unterliegen jedoch aufgrund des Sicherheitsmodells von Java (siehe Abschnitt 4.4) starken Restriktionen. Notwendige Operationen, wie Dateiverarbeitung und unbeschränkte Netzwerkkommunikation, können vom Applet nicht ausgeführt werden. Sie müssen ausgelagert werden und erfordern somit die Benutzung eines externen Servers.

Das Client-/Server-Modell bietet allerdings auch Vorteile. Durch serverbasiertes Arbeiten können die Daten, d.h. die Berichte, zentral gelagert und abgerufen werden und bei einer Installation von JReport auf einem Web-Server steht die Anwendung im gesamten Intranet bzw. Internet zur Verfügung. Nachteilig wirkt sich bei diesem Konzept vor allem der erhöhte Zeitaufwand aus, der für die Entwicklung zweier getrennter Applikationen notwendig ist.

#### 5.1.4 Der Applikations-Client

Ein beliebiger Java-fähiger Web-Browser bildet den Ausführungsrahmen des Applikations-Clients. Die JVM des Browsers führt das in eine Webseite eingebettete Java-Applet aus, das den Programmcode des Clients enthält. Es realisiert die Benutzerschnittstelle der gesamten Anwendung und ist somit für die optische Präsentation und die Verarbeitung der Benutzereingaben verantwortlich.



**Abbildung 19: Aufbau des Applikations-Clients**

Die Struktur des Clients, dargestellt in Abbildung 19, ist relativ einfach. Den Kern bilden die einzelnen Dialogmasken (siehe Abbildung 24 bis Abbildung 27), welche die Parameter für den zu generierenden Report erfassen. Die eingegebenen Daten werden mittels RPC-Funktionsaufrufen<sup>16</sup> auf Basis einer TCP-Socket-Schnittstelle an den Server übertragen. Das Ganze wird

<sup>16</sup> RPC - Remote Procedure Call: Eine Variante der Netzwerkprogrammierung, bei der Funktionen auf einem entfernten Server ausgeführt werden (siehe auch [STE96]).

von einer Clientsteuerung überwacht, die u.a. für die Initialisierung und die Ressourcenfreigabe bei Beendigung des Applets verantwortlich ist.

#### **5.1.4.1 Die Benutzeroberfläche**

Für die Gestaltung von grafischen Benutzeroberflächen (GUI - *Graphical User Interface*) verfügt Java über ein Set von Klassen unter dem Namen *Abstract Window Toolkit* (AWT). Es enthält die bekannten Standard-GUI-Komponenten wie Buttons, Checkboxes, Listboxen sowie Eingabefelder und stellt verschiedene Klassen für die Arbeit mit Fenstern, Ereignissen, Schriftarten und Grafiken bereit.

Auch die Oberfläche des Applikations-Clients beruht auf dem AWT. So sind die einzelnen Dialogmasken von der Klasse `Panel` abgeleitet - ein Allzweckcontainer für die Aufnahme visueller Komponenten und die Verarbeitung der damit anfallenden Ereignisse. Die Palette der GUI-Komponenten des AWT wurde durch einige Oberflächenelemente aus dem Entwicklungspaket „Visual Café 1.0“ von Symantec ergänzt. Dies betrifft die Komponenten:

- `TabControl` (Karteikartenreiter),
- `TreeView-Control` (Baumanzeige) und
- `MultiColumnListbox` (mehrspaltige Listbox),

allesamt Elemente, für die im AWT bisher kein entsprechendes Äquivalent zur Verfügung steht. Leider erwiesen sich die herstellerspezifischen Erweiterungen im Einsatz als problematisch. Aufgrund ihres frühen Entwicklungsstadiums enthielten sie Fehler, die jedoch mit Hilfe der in Abschnitt 5.2.2 beschriebenen Patches und Workarounds behoben werden konnten.

Insgesamt stellte sich die Programmierung der Benutzeroberfläche bezüglich der Portabilität der Anwendung als unerwartet schwierig heraus. Während alle anderen Bereiche von JReport sich durch den Einsatz von Java fast problemlos auf unterschiedliche Plattformen übertragen ließen, traten bei der Darstellung der Oberfläche doch zum Teil erhebliche Unterschiede auf.

Ursache sind die verschiedenen Größenverhältnisse der von Java unterstützten Schriftarten zwischen den einzelnen Betriebssystemen. Sie bedingen die unterschiedliche Darstellung der visuellen Komponenten des AWT. Als Folge kann die sonst übliche Positionierung der Oberflächenelemente mit Hilfe eines Koordinatensystems nicht angewendet werden. Es kann in Einzelfällen zu Überlappungen bzw. Überschneidungen der Komponenten kommen. Abhilfe bieten hier die in Java vorhandenen Layout-Manager. Sie steuern die Größe und Position der visuellen Komponenten innerhalb einer Containerklasse, z.B. einem `Panel`. Jeder Layout-Manager ordnet die Komponenten nach einem bestimmten Muster an. Die von Java unterstützten Manager sind in Tabelle 9 aufgeführt.

<b>Layout-Manager</b>	<b>Eigenschaften</b>
FlowLayout	ordnet die Komponenten einfach von links nach rechts an; beginnt eine neue Zeile, wenn der Platz für das Element nicht mehr ausreicht
GridLayout	stellt alle Komponenten in einem Gitterraster dar; jedes Element nimmt eine Zelle in diesem Gitter ein; alle Zellen sind gleich groß
BorderLayout	ordnet die Komponenten in fünf Bereichen an: Nord, Ost, Süd, West und Zentrum, wobei letzter Bereich sich über den gesamten verfügbaren Platz erstreckt
CardLayout	spezieller Manager, der die Komponenten in einem Kartenstapel verwaltet
GridBagLayout	sehr flexibler Manager, der die Komponenten ähnlich wie bei GridLayout in Zellen anordnet; allerdings können sich die Komponenten über mehrere Spalten und Zeilen erstrecken; weiterhin müssen nicht alle Spalten und Zeilen die gleiche Größe haben

**Tabelle 9: Layout-Manager in Java**

In den Dialogmasken wird vorrangig der GridBagLayout-Manager verwendet. Er bietet die größte Flexibilität für eine portable Anordnung der Oberflächenelemente. Leider ist er auch der komplexeste Layout-Manager; entsprechend aufwendig ist seine Programmierung. Vor allem, da die getesteten GUI-Builder (Visual Café und Kawa) die einzelnen Layout-Manager nicht bzw. nur mangelhaft unterstützen und meistens nur eine absolute Positionierung der Komponenten anbieten.

#### **5.1.4.2 Die Kommunikationsschnittstelle**

Der Datenaustausch zwischen Applikations-Client und Applikations-Server erfolgt mittels RPC-Aufrufen, eine Variante der Netzwirkommunikation der Firma Sun Microsystems aus den 80er Jahren (siehe RFC 1057). Bei dieser Form der Kommunikation ruft der Client wie gewohnt lokale Funktionen auf, die jedoch auf einem entfernten Server ausgeführt werden. Dazu implementiert der Client einen sogenannten *Stub*, der namens- und parametergleiche Äquivalente der aufzurufenden Funktionen enthält. Diese Stubfunktionen übernehmen die Parameter der gewünschten Funktion und setzen aus ihnen eine Netzwirknachricht für den Server zusammen. Im Server existiert ein entsprechendes Gegenstück des Client-Stubs, das die Nachrichten empfängt, die Parameter extrahiert und die eigentlich gewünschte Funktion ausführt. Die Ergebnisse dieses Aufrufs werden über den Stub an den Client zurückgeschickt.

Die Verwendung des RPC-Modells bietet verschiedene Vorteile bei der Entwicklung verteilter Anwendungen:

1. Die Programmierung gestaltet sich erheblich einfacher, da eine explizite Netzwerkkommunikation entfällt. Es werden, wie gewohnt, einfache Funktionsaufrufe implementiert.
2. Der Applikationsentwickler ist von der verwendeten Netzwerkschnittstelle weitgehend unabhängig. Die Details des benutzten Kommunikationsprotokolls und die notwendige Fehlerbehandlung sind Teil der RPC-Stubs und nicht der eigentlichen Anwendung.
3. Die Stubs auf Seiten des Clients und des Servers können bei Bedarf Konvertierungen der Funktionsparameter und Rückgabewerte vornehmen, um eventuell unterschiedliche Datenrepräsentationen auszugleichen.

Den RPC-Stub des Applikations-Clients realisiert die Klasse `ClientStub`. Sie implementiert derzeit neun Stubfunktionen, die bei späteren Änderungen von JReport erweitert werden können. Die einzelnen Funktionen sind in Tabelle 10 kurz erläutert.

<b>RPC-Funktion</b>	<b>Bedeutung</b>
<code>getServerVersion</code>	ermittelt die Versionsnummer des Applikations-Servers
<code>checkAddress</code>	überprüft die Gültigkeit einer IP-Adresse bzw. eines DNS-Namens
<code>checkAdressRange</code>	prüft einen Adressenbereich (siehe <code>checkAdress</code> )
<code>generateReport</code>	überträgt die Liste der abzufragenden Geräte und MIB-Objekte an den Applikations-Server und startet die Generierung eines Berichts
<code>getMIBTree</code>	ermittelt den aktuellen Hierarchiebaum aller vorhandenen MIB-Objekte des MIB-Managers
<code>loadMIB</code>	lädt eine MIB-Datei aus dem Verzeichnis „<Install-Verzeichnis>/mibs“ in den MIB-Manager des Applikations-Servers
<code>unloadMIB</code>	entfernt ein bereits geladenes MIB-Modul aus dem MIB-Manager des Applikations-Servers
<code>getMIBModuleList</code>	liefert die Namen der geladenen MIB-Module im MIB-Manager des Applikations-Servers zurück
<code>getMIBFileList</code>	ermittelt die Namen der Dateien im Verzeichnis „<Install-Verzeichnis>/mibs“

**Tabelle 10: Stubfunktionen des RPC**

Bei einem Aufruf einer Stubfunktion werden die Funktionsparameter zusammen mit der Funktionsnummer in ein Objekt der Klasse `RpcParameter` verpackt und an die Methode `serverCall` übergeben. Diese Methode kodiert die Elemente in einem Bytestream (siehe Abbildung 20) und übergibt sie als RPC-Nachricht an die darunterliegende Kommunikationsschicht. Anschließend wird auf eine Antwort des Servers bzw. eine Fehlermeldung der Kommunikationsschicht gewartet. Der zurückgelieferte Bytestream wird wieder in ein Objekt der Klasse `RpcParameter` verpackt und zur weiteren Verarbeitung an die aufrufende Stubfunktion gegeben.

RPC-Funktionsnr. (4 Bytes)	RPC-Status (4 Bytes)	funktionsabhängiges Datenfeld (0...x Bytes)
-------------------------------	-------------------------	--

**Abbildung 20: Aufbau einer RPC-Nachricht**

Die eigentliche Netzwerkkommunikation zwischen Applikations-Client und Applikations-Server basiert auf dem TCP-Protokoll und wird von der Klasse `ClientCommunication` abgewickelt. Das TCP-Protokoll ist ein verbindungsorientiertes und abgesichertes Kommunikationsprotokoll der TCP/IP-Suite ([STE96]), das sich für den zuverlässigen Datenaustausch zwischen Anwendungen eignet.

Um über TCP kommunizieren zu können, benötigen Client und Server eine Netzwerkverbindung. Dazu binden beide Programme einen *Socket* an einen Kommunikationsport und bauen zwischen ihnen eine bidirektionale Verbindung auf. Java stellt mit den Klassen `Socket` und `ServerSocket` im Package<sup>17</sup> `java.net` die entsprechenden Hilfsmittel zur Verfügung. Der Datenaustausch erfolgt durch einfaches Lesen und Schreiben auf dem zugehörigen `Socket`.

Diese Socketschnittstelle wird im Applikations-Client von der Klasse `ClientCommunication` realisiert. Sie baut beim Start eine Verbindung zum Applikations-Server auf, die bis zum Ende der Anwendersitzung bestehen bleibt. Dabei wird der Kommunikationsport des Clients dynamisch festgelegt, d.h. es wird eine aktuell freie Portnummer des Clientrechners verwendet. Die Portnummer des Servers muß dagegen statisch sein, damit eine Kommunikationsaufnahme vom Client zum Server möglich ist. Standardmäßig verwendet JReport die Nummer 10240 für den Kommunikationsport des Servers. Sie sollte in den meisten Fällen zur Verfügung stehen. Anderenfalls kann der Anwender über Parameter beim Server und beim Client eine neue freie Portnummer definieren.

---

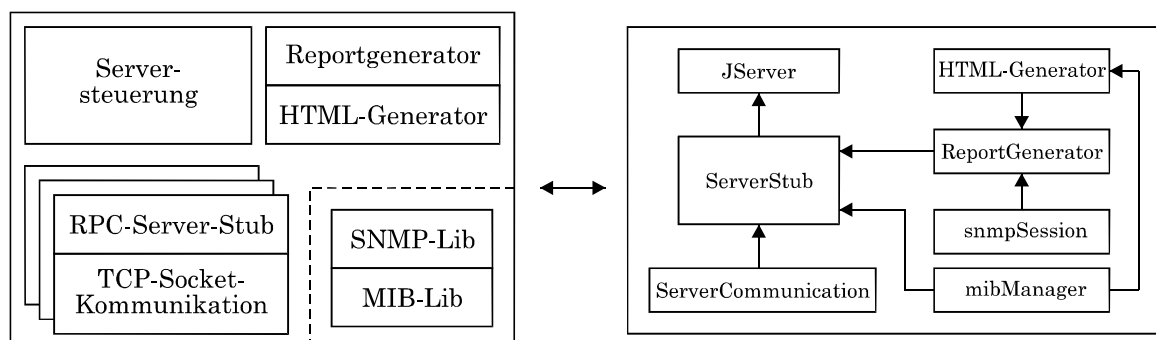
<sup>17</sup> Der Begriff „Package“ steht in Java für Softwarebibliothek bzw. -library. Damit können Klassen, die miteinander in Beziehung stehen, zusammengefaßt werden.

Java verfügt über verschiedene Packages, die Funktionen für bestimmte Arbeitsbereiche zur Verfügung stellen, z.B. `java.awt`, `java.graphics`, `java.net` u.a. Das sogenannte Default-Package `java.lang` wird automatisch benutzt; andere können mit der Anweisung „import <Package>;“ in den Quellcode eingebunden werden.

Das Kernstück der Klasse `ClientCommunication` bildet die Methode `transmit`. Sie nimmt den Bytepuffer mit den zu übertragenden RPC-Daten entgegen und schickt ihn mit einer entsprechenden Fehlerbehandlung über den Socket an den Applikations-Server. Anschließend wartet sie auf eine Antwort des Servers und gibt die empfangenen Daten an den Aufrufer der Methode im RPC-Stub zurück. Prinzipiell ist diese Verarbeitung mit Hilfe der in der Klasse `Socket` implementierten *Streams* relativ einfach. Die Daten werden beim Versand in den Sender-Stream des Sockets geschrieben und beim Empfang aus dem Empfänger-Stream gelesen. Leider stellte sich bei der Implementierung der Schnittstelle heraus, daß die Netzwerk-API in der Java-Version 1.0.2 in bestimmten Situationen ein ungewöhnliches und zum Teil auch fehlerhaftes Verhalten aufweist. Für die Lösung der aufgetretenen Probleme war der Einsatz zweier Workarounds erforderlich, die im Abschnitt 5.2.2 beschrieben werden.

### 5.1.5 Der Applikations-Server

Der Applikations-Server ist der Dienstbringer der Anwendung JReport. Er ist aufgrund der Sicherheitsrestriktionen für Java-Applets notwendig und übernimmt stellvertretend alle diejenigen Aufgaben, die der Applikations-Client nicht selbst ausführen kann. Dazu zählen u.a. die SNMP-Kommunikation mit den gewünschten Netzkomponenten und die Generierung der entsprechenden Berichte. Der Applikations-Server ist MultiClient-fähig und kann somit seine Dienste mehreren Klienten zur Verfügung stellen. Dazu implementiert er mehrere Threads, von denen jeweils einer für einen Client zuständig ist. Der Server basiert auf einer Java-Applikation, die in einer einfachen Textkonsole abläuft. Die wesentlichen Bestandteile seiner Struktur werden in Abbildung 21 dargestellt und nachfolgend erläutert.



**Abbildung 21: Aufbau des Applikations-Servers**

Die Klasse `JServer` realisiert die Steuerung des gesamten Applikations-Servers. Sie enthält die öffentliche Startfunktion `main` der Anwendung, welche die Initialisierung der restlichen Bestandteile des Servers übernimmt. Den Kern der Klasse stellt der sogenannte *Server-Thread* dar, der auf die Kommunikationsanfragen neuer Klienten wartet. Dazu implementiert er mit einem Objekt der Klasse `ServerSocket` den serverseitigen Endpunkt einer TCP-Verbindung. Dieser, auch als *Listener* bezeichneten Teil des Threads, wartet am Kommunika-



tionsport 10240 auf eingehende Verbindungsanforderungen. Dabei sollte die Portnummer i.allg. frei zur Verfügung stehen. Sie kann jedoch bei Bedarf mit dem Startparameter „-port XXX“ durch den Anwender geändert werden. Die Verbindungsanforderung des neuen Clients löst die Erzeugung eines neuen Sockets aus, der an eine freie Portnummer des Serverrechners gebunden wird. Jede weitere Kommunikation zwischen Server und Client erfolgt über diesen neuen Socket, so daß die ursprüngliche Portnummer 10240 für Verbindungsanforderungen weiterer Clients zur Verfügung steht.

Für jeden neuen Applikations-Client legt der Server-Thread einen eigenen Server-Stub an (realisiert in der Klasse `ServerStub`), der die weitere RPC-Kommunikation abwickelt. Jeder Stub basiert auf einem Thread, so daß die Klienten vollkommen getrennt und unabhängig voneinander bedient werden. Der Thread endet, wenn der Client seine Verbindung zum RPC-Server-Stub abbaut. Der Server-Stub stellt das Gegenstück zum Stub des Applikations-Clients dar. Er implementiert die in Tabelle 10 aufgelisteten RPC-Funktionen.

Die für die Anwendung wichtigste Funktion, der Reportgenerator, wird zusammen mit seiner Ausgabefunktion, dem HTML-Generator, in den Abschnitten 5.1.5.2 und 5.1.5.3 erläutert. Es bleiben noch die beiden Klassen `snmpSession` und `mibManager`. Sie stellen die öffentlichen Schnittstellen der beiden Softwarebibliotheken SNMP-Lib und MIB-Lib dar und werden in den Abschnitten 5.1.6 und 5.1.7 näher untersucht.

#### **5.1.5.1 Die Kommunikationsschnittstelle**

Die Kommunikation zwischen Client und Server wird von den Klassen `ServerStub` und `ServerCommunication` abgewickelt. Beide Klassen gleichen in ihrer Struktur denen auf der Seite des Clients, d.h. `ServerStub` übernimmt die Ausführung der RPC-Funktionen und `ServerCommunication` den eigentlichen Datentransfer vom und zum Client.

Der Kern der Klasse `ServerCommunication` ist ein Thread, der auf eingehende Kommunikationsmeldungen vom Client wartet. Die Daten werden blockweise empfangen (siehe Probleme bei der Kommunikation mittels Streams, Abschnitt 5.1.4.2) und die resultierende RPC-Nachricht an die Methode `serverCall` des RPC-Stubs übergeben. Diese dekodiert die empfangene Nachricht und führt die gewünschte RPC-Funktion aus. Dazu wird den RPC-Funktionsnummern eine reale Methode des Server-Stub zugeordnet und ausgeführt. Die Ergebnisse des Funktionsaufrufs werden wieder in eine RPC-Nachricht verpackt (siehe Abbildung 20) und an die Kommunikationsschicht übergeben. Diese sendet die Antwort an den Client zurück.

Die meisten der in Tabelle 10 aufgeführten Funktionen sind relativ einfach und bedürfen keiner weiteren Erläuterung. Interessant ist jedoch die RPC-Funktion *generateReport*. Hinter ihr verbirgt sich die eigentliche Funktionalität der Anwendung JReport, d.h. die Abfrage von SNMP-Informationen und die Verarbeitung der gewonnenen Daten zu einem Bericht. Diese

Funktionalität wird von der Klasse `ReportGenerator` realisiert, deren Beschreibung im nächsten Abschnitt 5.1.5.2 folgt.

#### 5.1.5.2 Der Reportgenerator

Der Reportgenerator, implementiert in der Klasse `ReportGenerator`, realisiert den eigentlichen Kern der gesamten Anwendung JReport. Anhand der Anwendereingaben im Applikations-Client werden von ihm die gesuchten SNMP-Informationen von den gewünschten Netzkomponenten abgefragt, aufbereitet und in einem Bericht im Dokumentenformat HTML zusammengefaßt.

Die Grundlage seiner Arbeit bilden zwei Listen, die ihm vom Applikations-Client übermittelt werden. Sie enthalten die Netzwerkadressen und SNMP-Community-Namen der abzufragenden Geräte sowie OIDs (*Object Identifier*) der abzufragenden SNMP-Objekte. Vor dem eigentlichen Prozeß der Informationsermittlung müssen diese beiden Listen zu einer speziellen Abfrageliste aufbereitet werden. Dies geschieht in der Methode `prepareQueryList`. Nach dieser Vorverarbeitung enthält die Liste jeweils einen Eintrag pro abzufragendes Gerät. Die Strukturen der Listenelemente enthalten Felder für die Aufnahme der Netzadresse, der SNMP-Community und des Abfragestatus. Weiterhin existiert pro Eintrag ein Verweis auf eine SNMP-Objektliste, welche die abzufragenden OIDs und deren Ergebnisse aufnimmt.

Mit der so präparierten Abfrageliste wird die Datensammlung gestartet, bei der die Funktionen der SNMP-Lib (siehe Abschnitt 5.1.6) zum Einsatz kommen. Die Library übernimmt alle Aufgaben, die bei der Arbeit mit dem SNMP-Protokoll anfallen. Dazu zählen u.a. das Kodieren und Dekodieren von SNMP-Objekten auf der Basis der SNMP-Datentypen, die Zusammenstellung von SNMP-Nachrichten für die Ausführung einer bestimmten SNMP-Operation sowie der Versand und Empfang der zugehörigen SNMP-PDUs. Die öffentliche Schnittstelle dieser Softwarebibliothek bildet die Klasse `snmpSession`. Sie implementiert ein vollständiges Sessionmanagement mit einer entsprechenden Fehlerbehandlung und reduziert so den Programieraufwand im Reportgenerator für den Versand von SNMP-Anfragen erheblich.

Der eigentliche Prozeß der Informationsgewinnung gestaltet sich relativ einfach. Die Abfrageliste wird in einer Schleife durchlaufen. Für jedes Gerät werden die OIDs der gewünschten SNMP-Objekte zusammen mit der Empfängeradresse und der SNMP-Community an die SNMP-Lib übergeben. Diese konstruiert daraus eine SNMP-PDU und schickt sie an die angegebene Netzkomponente. Die Antwort des Gerätes wird dekodiert und die Ergebnisse der Anfrage beim zugehörigen Element der Abfrageliste eingetragen. Sollte bei einem Gerät ein schwerwiegender Fehler auftreten (z.B. Timeout), so wird die Verarbeitung abgebrochen und beim nächsten Geräteeintrag fortgesetzt.

Der Reportgenerator verschickt allerdings pro Anfrage immer nur eine OID eines SNMP-Objektes. Sollen also mehrere OIDs abgefragt werden, so werden sie einzeln in Form von meh-

rerer SNMP-PDUs verschickt. Aus Sicht der Netzwerkperformance wäre es sicherlich günstiger, die OIDs in einer PDU zusammenzufassen. Dies kann jedoch an der jeweiligen Implementierung des SNMP-Agenten scheitern, da sie die Länge einer SNMP-Nachricht begrenzen können. Der Agent muß gemäß dem SNMPv1-Standard nur Nachrichten bis zu einer Länge von 484 Bytes verarbeiten. Größere PDUs kann er mit einem Fehler *tooBig* abweisen. Zusätzlich existieren immer noch SNMP-Agenten, die lediglich eine OID pro PDU verarbeiten können. Die Berücksichtigung all dieser Sonderfälle würde dann eine erheblich aufwendigere Programmlogik erfordern.

Der Reportgenerator verwendet für die Abfrage der Geräteinformationen die SNMP-Operation „GetNextRequest“ (siehe Abschnitt 3.5.3). Sie eignet sich sowohl für die Erfassung einfacher (singulärer) SNMP-Objekte als auch für die Abfrage multipler Objektinstanzen, wie sie bei Tabellen auftreten. Letzteres bezieht sich auf OIDs, die eine Tabellenspalte referenzieren. Dann können mit wiederholt ausgeführten Operationen vom Typ „GetNextRequest“ alle zugehörigen Zeilenwerte ermittelt werden. Die Operation erfordert jedoch eine etwas umfangreichere Auswertung, um sicherzustellen, ob es sich beim zurückgelieferten SNMP-Objekt um das eigentlich gesuchte handelt. Dazu werden die OIDs der beiden SNMP-Objekte verglichen und folgende Fallunterscheidung durchgeführt:

1. Die abgesandte OID muß vollständig in der zurückgelieferten OID enthalten sein. Wenn das nicht der Fall ist, siehe weiter Punkt 4.
2. Falls die letzte SubID der zurückgelieferten OID eine Null ist und ansonsten alle Stellen der beiden OIDs übereinstimmen, war die Abfrage erfolgreich. Es handelt sich um ein einfaches (singuläres) SNMP-Objekt.
3. Falls die letzte(n) SubID(s) der zurückgelieferten OID verschieden von Null sind, während der Rest mit der ursprünglich abgesandten OID übereinstimmt, so war die Abfrage ebenfalls erfolgreich. Es handelt sich um die Objektinstanz einer Tabellenspalte. Um auch die restlichen Zeilenwerte der Tabellenspalte zu ermitteln, muß die Operation „GetNextRequest“ mit der zurückgelieferten OID wiederholt werden.
4. Die zurückgelieferte OID stimmt mit der abgesandten nicht überein. Falls die letzte Operation eine Tabellenspalte betraf, so wurde jetzt das Ende dieser Spalte erreicht. Es wurden alle zugehörigen Objektinstanzen ermittelt und die Abfrage kann erfolgreich beendet werden. Ansonsten kennt der Agent des abgefragten Gerätes das gesuchte SNMP-Objekt nicht und liefert statt dessen das nächste in der Ordnung folgende Objekt. Die Abfrage ist somit fehlgeschlagen.

Nach der vollständigen Abarbeitung der Abfrageliste stehen in den Listenelementen die ermittelten SNMP-Informationen für die einzelnen Netzkomponenten. Aus ihnen wird der eigentliche Report generiert. Diese Funktionalität ist jedoch nicht Teil des Reportgenerators. Er bindet eine spezielle Ausgabeverarbeitung ein und übergibt ihr die „gefüllte“ Abfrageliste. Derzeit steht dafür nur der HTML-Generator zur Verfügung. Er bereitet die gewonnenen Informatio-

nen zu einem Bericht im Dokumentenformat HTML auf. Denkbar wären jedoch auch andere Ausgabeformate, z.B. für die Unterstützung einer Tabellenkalkulation oder einer Datenbank-schnittstelle. Diese Erweiterungen ließen sich in einer späteren Version von JReport mit geringen Änderungen realisieren.

#### **5.1.5.3 Der HTML-Generator**

Der HTML-Generator, implementiert in der Klasse `HTMLGenerator`, realisiert die Datenaufbereitung und -ausgabe für den Reportgenerator. Dabei generiert seine Ausgabefunktion aus den Informationen der übergebenen Abfrageliste (siehe Reportgenerator) einen Bericht im Dokumentenformat HTML.

Die erzeugten Reportdateien werden im Verzeichnis „<Install-Verzeichnis>/reports“ abgelegt. Die jeweiligen Dateinamen setzen sich aus Datum und Uhrzeit zum Zeitpunkt der Erstellung zusammen. Dies vermeidet Konflikte in der Namensgebung zwischen evtl. parallel laufenden Server-Stubs. Um dem Anwender den Umgang mit kryptischen Dateinamen zu ersparen, pflegt der HTML-Generator zusätzlich eine Indexdatei namens „report\_index.html“. Sie enthält eine Liste von URLs, welche auf die einzelnen Reportdateien verweisen. Jeder dieser Links ist mit der Titelüberschrift des Berichts gekennzeichnet (siehe Abbildung 28). Neue Reports werden in der Reihenfolge ihrer Erstellung an das Ende der Indexliste angefügt. So kann der Anwender sie in chronologischer Ordnung abrufen.

Der eigentliche Verarbeitungsprozeß gestaltet sich wie folgt: Der HTML-Generator öffnet einen Ausgabe-Stream in eine Datei und druckt die Informationen, eingeschlossen in entsprechende HTML-Kommandos, in diesen Stream. Dabei wird die Abfrageliste in einer Schleife abgearbeitet und für jeden Geräteeintrag eine Tabelle mit jeweils zwei Spalten erstellt (siehe Abbildung 29). Auf der linken Seite der Tabelle erscheinen die Namen der gesuchten SNMP-Objekte und auf der rechten die ermittelten Werte bzw. eine Fehlermeldung, falls Probleme bei der Abfrage aufgetreten sind. Vor der Ausgabe findet zum Teil eine Konvertierung der ermittelten Werte mit Hilfe der MIB-Library (siehe Abschnitt 5.1.7) statt. Der Report schließt mit einer Übersicht, in der den Objektnamen die entsprechenden OIDs zugeordnet werden.

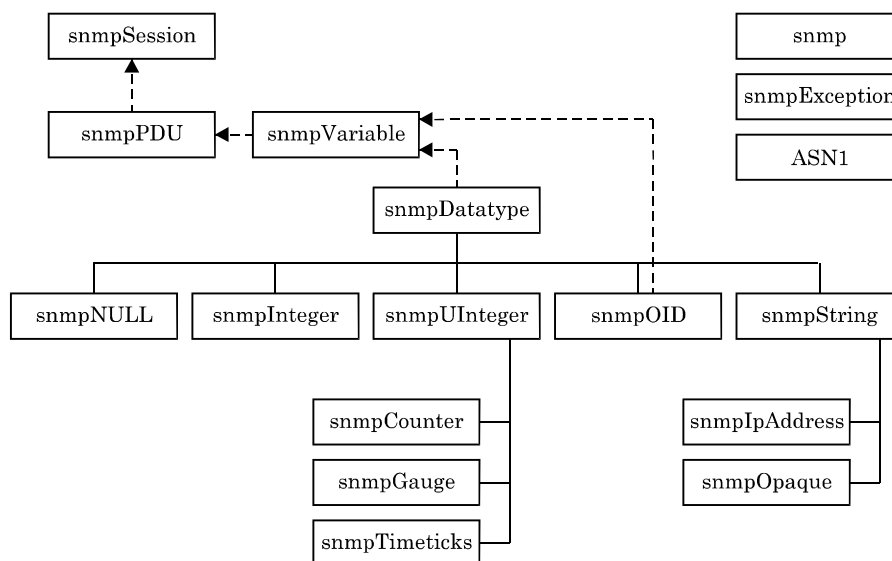
#### **5.1.6 Die SNMP-Library**

Die SNMP-Lib ist eine Java-Softwarebibliothek (Package `snmp`), die ein umfangreiches Set an Klassen und Methoden für die Nutzung des Netzwerkmanagementprotokolls SNMPv1 zur Verfügung stellt. Sie übernimmt alle Aufgaben, die bei der Arbeit mit diesem Protokoll anfallen. Dazu zählen u.a. das Kodieren und Dekodieren von SNMP-Objekten auf der Basis von SNMP-Datentypen, die Zusammenstellung von SNMP-Nachrichten für die Ausführung bestimmter SNMP-Operationen sowie der Versand und Empfang von SNMP-PDUs. Derzeit un-

terstützt die Library alle SNMPv1-Operationen (siehe Abschnitt 3.5), wobei allerdings das Kommando TRAP mangels Verwendung noch nicht getestet wurde.

Die Grundlage der SNMP-Library ist die frei verfügbare SNMP-Implementierung der Carnegie Mellon Universität [CMU12]. Sie hat sich im Einsatz bewährt und dient als Basis zahlreicher SNMP-Applikationen. Ohne den Rückgriff auf diese Implementierung wäre die ohnehin recht umfangreiche Entwicklungszeit der Anwendung JReport beträchtlich angewachsen, da in den zur Verfügung stehenden Literaturquellen keinerlei Hinweise für eine praktische Umsetzung des SNMPv1-Standards gegeben werden. Die Library der Carnegie Mellon Universität diente somit als maßgebliche Wissensquelle.

Die beiden Implementierungen unterscheiden sich jedoch erheblich. Während die Library der Carnegie Mellon Universität auf Funktionen der Programmiersprache C basiert, realisiert die SNMP-Lib einen objektorientierten Ansatz in Java. Damit lassen sich verschiedene Probleme eleganter lösen. So ist die Implementierung der SNMP-Datentypen wesentlich einfacher sowie leichter erweiterbar. Die Klassenstruktur der SNMP-Lib wird in Abbildung 22 dargestellt. Sie zeigt in Ansätzen die Verknüpfungen und Ableitungen der einzelnen Klassen. Die Tabelle 11 beschreibt kurz den zugehörigen Aufgabenbereich.



**Abbildung 22: Struktur der SNMP-Library**

Die SNMP-Library bietet mit der Klasse `snmpSession` eine einfache Schnittstelle, die eine schnelle Integration in Anwendungen ermöglicht. Diese Klasse realisiert ein vollständiges Sessionmanagement und übernimmt so die Verwaltung der Kommunikation zwischen Anwendung und SNMP-Agent. Damit kann der Versand der Managementnachrichten in Form der SNMP-PDUs mit wenigen Programmanweisungen ausgeführt werden. Die Klasse `snmpSession` überwacht den Transfer und führt bei Bedarf die notwendige Fehlerbehandlung durch, z.B. ein erneutes Absenden bei einem Timeout-Fehler.

Klassenname	Klassenbeschreibung
snmpSession	Öffentliche Schnittstelle der SNMP-Lib. Implementiert die Verwaltung der Anwendersitzungen (Sessionmanagement).
snmpPDU	Dient der Kodierung und Dekodierung von SNMP-PDUs für die Operationen GetRequest, GetNextRequest, SetRequest und Trap.
snmpVariable	Implementierung einer SNMP-Variable (auch SNMP-Objekt genannt), bestehend aus einer SNMP-OID und einem beliebigen SNMP-Datentyp.
snmpDatentyp	Ein abstrakter SNMP-Datentyp, der die Grundlage für alle anderen Datentypen (NULL, OID, String, ...) bildet. Jeder dieser abgeleiteten SNMP-Datentypen muß mindestens die Methoden <code>build()</code> , <code>parse()</code> , <code>getType()</code> und <code>toString()</code> implementieren.
snmpNULL	Implementiert den ASN.1-Grunddatentyp NULL.
snmpInteger	Implementiert den ASN.1-Grunddatentyp INTEGER.
snmpOID	Implementiert den ASN.1-Grunddatentyp OBJECT IDENTIFIER.
snmpString	Implementiert den ASN.1-Grunddatentyp OCTET STRING. Vorlage für die Klassen <code>snmpIpAddress</code> und <code>snmpOpaque</code> .
snmpIpAddress	Verfeinerter SMI-Datentyp <code>IpAddress</code> .
snmpOpaque	Verfeinerter SMI-Datentyp <code>Opaque</code> .
snmpUnsignedInteger	Vorzeichenlose Variante des ASN.1-Typs INTEGER. Vorlage für die Klassen <code>snmpCounter</code> , <code>snmpGauge</code> und <code>snmpTimeTicks</code> .
snmpCounter	Verfeinerter SMI-Datentyp <code>Counter</code> .
snmpGauge	Verfeinerter SMI-Datentyp <code>Gauge</code> .
snmpTimeTicks	Verfeinerter SMI-Datentyp <code>TimeTicks</code> .
ASN1	Beinhaltet Methoden für die Kodierung und Dekodierung von ASN.1-Datentypen von und nach BER.
snmp	Beinhaltet alle Konstantendefinitionen, z.B. für Kommandos, Fehlermeldungen, Defaultwerte u.ä.
snmpException	Definiert eine Ausnahme vom Typ „SNMP-Exception“. Sie wird ausgelöst, wenn ein Fehler in der SNMP-Lib auftritt.

**Tabelle 11: Klassen der SNMP-Library**

Wie eine Integration der SNMP-Lib in eine Anwendung aussehen kann, zeigen die kleinen kommandozeilenbasierten Tools, welche die typischen SNMP-Operationen ausführen. Sie befinden sich zusammen mit den Quelltexten im Verzeichnis „<Install-Verzeichnis>/snmptools“. Diese Tools wurden u.a. während der Entwicklungsphase für Tests an der SNMP-Lib benutzt.

### 5.1.7 Die MIB-Library

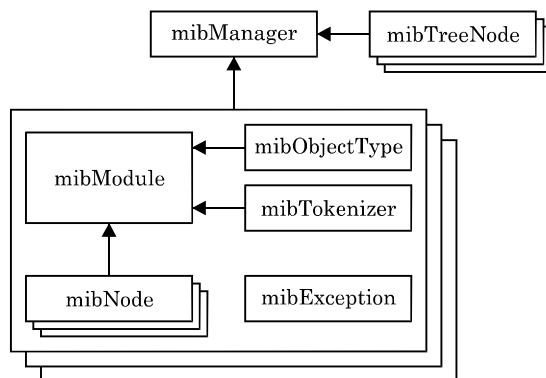
Die MIB-Library ist eine Java-Softwarebibliothek (Package `mib`), die ein Set von Klassen für den Umgang mit der Management Information Base zur Verfügung stellt. Sie implementiert einen flexiblen Parser für die Analyse von MIB-Dateien sowie eine Verwaltung der eingelesenen MIB-Module. So können neben der Standard-MIB [RFC1213] auch zahlreiche hersteller-spezifische MIB-Dateien eingelesen werden. Der Anwendungsbereich der MIB-Lib erstreckt sich vorrangig auf die Unterstützung der Benutzer von Netzwerkmanagementtools. Mit den Lib-Funktionen können abgefragte SNMP-Informationen aufbereitet und für den Anwender verständlich gemacht werden. Eine dieser wesentlichen Funktionen ist die wechselseitige Konvertierung von OIDs (*Object Identifier*) in zugehörige MIB-Objektnamen, z.B.:

`{1.3.6.1.2.1.1.1}` nach `{iso.org.dod.internet.mgmt.mib-2.system.sysDescr}`.

Weiterhin können Indexnummern, wie sie z.B. bei der Abfrage von Schnittstellentypen auftreten, durch klar lesbare Bezeichnungen ersetzt werden (`ifType(6): ethernet-csmacd`). Für die Applikation JReport ist noch eine weitere Funktion von wesentlicher Bedeutung. Die Library kann den aktuellen Hierarchiebaum aus den MIB-Objekten der MIB-Module erstellen. Entsprechend aufbereitet, wird dieser Baum dem Anwender im TreeView-Control des Applikations-Clients präsentiert (siehe Abbildung 26). Er gestattet dem Anwender die flexible Navigation in den Strukturen der MIB und die komfortable Auswahl der gewünschten MIB-Objekte.

Die Grundlage für die Entwicklung der MIB-Library bildete wieder die frei verfügbare SNMP-Implementierung der Carnegie Mellon Universität [CMU12]. Sie enthält neben den Funktionen für das Netzwerkmanagementprotokoll SNMPv1 auch einen einfachen Parser, der das Einlesen einer MIB-Datei gestattet. Dieser Parser ist allerdings noch im Entwicklungsstadium. Seine Funktionalität ist bescheiden und seine Vorgehensweise relativ inflexibel. So ist er auf eine feste Reihenfolge bestimmter Strukturen im MIB-Modul angewiesen und kann mit gelegentlich auftretenden Abweichungen nicht umgehen. Im Ergebnis liest er die Standard-MIB [RFC1213]; scheitert allerdings häufig an herstellerspezifischen MIB-Dateien. Trotzdem stellt dieser Teil der Implementierung der Carnegie Mellon Universität eine wertvolle Wissensquelle dar, aus der zahlreiche Anregungen für die Entwicklung der MIB-Library entnommen wurden. Mit ihrer Hilfe konnte die grundlegende Vorgehensweise für die Analyse der Informationen eines MIB-Moduls geklärt werden.

Die Abbildung 23 zeigt die derzeitige Struktur der MIB-Library. Dargestellt werden die Beziehungen zwischen einzelnen Klassen, zu deren Aufgabenbereich die Tabelle 12 eine kurze Beschreibung liefert.



**Abbildung 23: Struktur der MIB-Library**

Klassenname	Klassenbeschreibung
mibManager	Bildet die öffentliche Schnittstelle der MIB-Lib. Die Klasse implementiert einen Manager für die Verwaltung der MIB-Module.
mibModule	Realisiert den eigentlichen MIB-Parser, der MIB-Dateien analysiert und die gewonnenen Daten abspeichert. Nach der Auswertung repräsentiert die Klasse die Informationen eines MIB-Moduls.
mibObjectType	Ein spezieller Teil des MIB-Parsers, der das Statement „OBJECT-TYPE“ (die Definition eines MIB-Objektes) analysiert. Aus den gewonnenen Informationen wird ein Objekt der Klasse mibNode erzeugt.
mibTokenizer	Implementiert einen Tokenizer, der die MIB-Datei in ihre Schlüsselwörter ( <i>tokens</i> ) zerlegt und Kommentarzeilen herausfiltert. Vereinfacht wesentlich die weitere Analyse des Parsers.
mibNode	Nimmt die Informationen (OID, Objektname, Objekttyp, Status, Beschreibung) eines eingelesenen MIB-Objektes auf. Alle vorhandenen Knoten bilden zusammen den Hierarchiebaum der MIB.
mibTreeNode	Ein „visueller“ Knoten des TreeView-Controls im Client, der temporär aus den Informationen der Klasse mibNode generiert wird.
mibException	Definiert eine Ausnahme vom Typ „MIB-Exception“. Sie wird ausgelöst, wenn ein Fehler in der MIB-Lib auftritt.

**Tabelle 12: Klassen der MIB-Library**



Die Klasse `mibManager` stellt die Anwendungsschnittstelle der MIB-Library dar. Sie realisiert einen Manager für die Verwaltung von MIB-Dateien und ermöglicht die Abfrage von Informationen bereits analysierter MIB-Module. Diese Daten können für die Aufbereitung abgefragter SNMP-Informationen benutzt werden und so die Lesbarkeit für Anwender erhöhen. Der Manager verwaltet die analysierten MIB-Module in Objekten der Klasse `mibModule`, die auf der Grundlage einer MIB-Datei neu erstellt und bei Nichtgebrauch wieder aus dem Manager entfernt werden können.

Die Klasse `mibModule` implementiert den eigentlichen Parser der Library, der eine MIB-Datei einliest, in ihre Bestandteile zerlegt und die ermittelten Informationen in Strukturen der Klasse `mibNode` abspeichert. Dabei versetzen verschiedene Mechanismen den Parser in die Lage, neben der Standard-MIB [RFC1213], auch zahlreiche herstellerspezifische und somit proprietäre MIB-Dateien erfolgreich zu analysieren. So verfügt er über eine zur Laufzeit erweiterbare Schlüsselwortliste, mit der die Verarbeitung einfacher Substitutionsausdrücke möglich ist, z.B.:

```
-- textual conventions
DisplayString = OCTET STRING (SIZE (0..255))
MacAddress    = OCTET STRING (SIZE (6)).
```

Weiterhin wertet der Parser das Statement `IMPORTS` aus und kann so erweiterte SMI-Datentypen und definierte MIB-Objekte aus anderen MIB-Modulen benutzen. Allerdings ist er an dieser Stelle auf die Mithilfe des Anwenders angewiesen, da er nur Informationen aus bereits geladenen MIB-Modulen importieren kann. Sollte also an dieser Stelle ein Fehler auftreten, so weist er den Anwender i.allg. darauf hin, daß ein bestimmtes MIB-Modul als Importquelle nicht verfügbar ist und erst noch geladen werden muß. Unterstützung erhält der Parser in der Klasse `mibModule` auch von anderer Seite. So realisiert die Klasse `mibTokenizer` einen speziellen Stream, der die Eingabedaten der MIB-Datei vorverarbeitet und z.B. die Kommentarzeilen bereits herausfiltert. So werden nur noch die relevanten Schlüsselwörter (*tokens*) an den Parser übergeben. Zusätzlich ist ein Teil des Parsers aus Gründen der Übersichtlichkeit in die Klasse `mibObjectType` ausgelagert. Sie übernimmt die Verarbeitung des Makros `OBJECT-TYPE`, welches die Definition eines MIB-Objektes beinhaltet. Nach Abschluß aller Analysen repräsentiert jede Instanz der Klasse `mibModule` die Daten eines MIB-Moduls.

Der Parser der MIB-Library stellt in der derzeitigen Form einen Kompromiß zwischen der Forderung nach erfolgreicher Verarbeitung möglichst vieler unterschiedlicher MIB-Dateien und dem dafür notwendigen Entwicklungsaufwand dar. So gibt es noch genügend Situationen, die der Parser nicht oder nur mit Hilfe des Anwenders bewältigt. Einige Elemente der SMI (siehe Abschnitt 3.4.3) sind aus Gründen der derzeitigen Nichtnotwendigkeit bzw. wegen ihres seltenen Gebrauchs nicht in der MIB-Library realisiert. Dies trifft z.B. für das Makro `TRAP-TYPE` zu. Es wird in der aktuellen Version nicht ausgewertet, da Traps für die Arbeit mit der

Anwendung JReport nicht notwendig waren. Auch umfangreiche Substitutionsausdrücke, die über das obige Beispiel hinausgehen, können derzeit noch nicht verarbeitet werden. Sie treten jedoch nur relativ selten auf und können notfalls durch ein manuelles „Suchen und Ersetzen“ in einem Texteditor beseitigt werden.

## **5.2 Die implementierte Softwarelösung**

### **5.2.1 Systemanforderungen**

Die Anwendung JReport stellt relativ geringe Ansprüche an die Arbeitsplattform. Voraussetzung ist ein installiertes Netzwerkinterface mit einem konfigurierten TCP/IP-Protokollstack. Durch die Verwendung von Java kann prinzipiell jedes Rechnersystem verwendet werden, für das eine entsprechende Laufzeitumgebung zur Verfügung steht. Bei einigen Betriebssystemen, z.B. IBM OS/2 Warp 4 und Linux 2.0, sind solche JREs (*Java Runtime Environment*) bereits integriert, für andere kann diese Umgebung nachinstalliert werden. Ansonsten wird lediglich noch ein Java-fähiger Web-Browser benötigt, deren bekanntesten Vertreter der Netscape Navigator bzw. Communicator und der Microsoft Internet Explorer sind.

### **5.2.2 Patches und Workarounds**

Während der Implementierung der Anwendung JReport mit Hilfe des JDK 1.0.2 von Sun unter dem Betriebssystem Microsoft Windows NT 4.0 kam es zu Situationen, in denen das Java-Laufzeitsystem ein ungewöhnliches und zum Teil auch fehlerhaftes Verhalten aufwies. Die festgestellten Probleme traten häufig nur bei einzelnen Systemen auf und konnten unter anderen Testumgebungen nicht nachvollzogen werden. Die meisten der Fehler sind Sun bereits bekannt und in der weiterentwickelten Java-Version 1.1 beseitigt. Doch für deren Lösung unter der Java-Version 1.0.2 kann man i.allg. nur auf Patches und Workarounds anderer Java-Entwickler zurückgreifen.

Zwei dieser Probleme wurden bei der Entwicklung der Kommunikationsschnittstelle zwischen Applikations-Client und Applikations-Server in Verbindung mit der Netzwerk-API festgestellt. Die Fehler scheinen im plattformabhängigen Teil der jeweiligen Java-Laufzeitumgebung zu liegen und treten u.a. in der Win32-Implementierung (d.h. Windows NT / 95) des JDK 1.0.2 von Sun auf. Das erste Problem betrifft die Klasse `InetAddress`. Sie wird benötigt, um einen *String* mit einer IP-Adresse bzw. einem Hostnamen in ein in Java verwendbares Objekt einer Internetadresse zu konvertieren. Dabei lösen Hostnamen, denen per DNS-Lookup<sup>18</sup> keine gül-

---

<sup>18</sup> DNS - Domain Name System: Ein Dienst des Internets, der die Beziehung zwischen Hostnamen und numerischen IP-Adressen herstellt. Unter DNS-Lookup wird die Anfrage an einen DNS-Server verstanden, die für einen Hostnamen die zugeordnete(n) IP-Adresse(n) ermittelt (siehe auch [STE96]).

tige IP-Adresse zugeordnet werden kann, völlig korrekt die Ausnahme `unknownHostException` aus. Der eigentliche Fehler tritt bei der Verwendung numerischer IP-Adressen auf. Hier versucht die Klasse `InetAddress` einen umgekehrten (*reverse*) DNS-Lookup auszuführen, um der IP-Adresse einen Hostnamen zuordnen zu können. Dies schlägt jedoch unter den obengenannten Umgebungen fehl, wenn kein DNS-Server konfiguriert wurde bzw. wenn der DNS-Server die Auflösung in umgekehrter Richtung nicht unterstützt. Die Folge ist ebenfalls eine Ausnahme `unknownHostException`. Im Normalfall sollte jedoch die Arbeit mit numerischen IP-Adressen immer möglich sein. Abhilfe bietet ein Workaround<sup>19</sup> von Peter Parnes. Seine neu implementierte Klasse `InetAddress` ersetzt die vorhandene im JDK 1.0.2.

Das zweite Problem betrifft die Kommunikation zwischen Applikations-Client und Applikations-Server über TCP und tritt bei der Datenübertragung mit Hilfe der Klasse `Socket` auf. Beim Versand größerer Datenmengen (ab ca. 13 KB) kommt es beim Empfänger zu Unstimmigkeiten und Fehlern. So weicht die Zahl der empfangenen Bytes von der der abgesendeten ab, und der Inhalt der empfangenen Nachricht stimmt nicht mit dem der gesendeten überein. Die Fehler scheinen von der Pufferverwaltung der Klasse `BufferedInputStream` bzw. `BufferedOutputStream` in Java verursacht zu werden. Sie treten vorrangig auf, wenn Client und Server auf dem gleichen Rechner ablaufen. Dies könnte u.a. dadurch bedingt sein, daß der Empfänger die Daten nicht schnell genug aus dem Stream des Sockets liest. Da der Versand kleinerer Nachrichten völlig problemlos funktioniert, implementiert die Klasse `ClientCommunication` des Applikations-Clients folgenden Workaround: Der Versand der Daten erfolgt über die Streams der Socketschnittstelle in kleinen Blöcken mit der Größe von 1 KB. Jeder gesendete Block muß erst von der Gegenseite bestätigt werden, bevor der nächste abgeschickt wird. Mit diesem Verfahren klappt auch die Übertragung sehr großer Datenmengen zwischen dem Applikations-Client und dem Applikations-Server. Es bleibt jedoch nur eine Notlösung, da normalerweise Fehler dieser Art durch die Verwendung des TCP-Protokolls explizit ausgeschlossen sein sollten.

Aber auch sonst traten bei der Implementierung von JReport unvorhergesehene Probleme auf. So erwiesen sich die herstellereigenen AWT-Erweiterungen aus dem Entwicklungspaket „Visual Café 1.0“ von Symantec, die für die Gestaltung der Benutzeroberfläche eingesetzt wurden, als fehlerhaft. Dies betrifft die Komponenten:

- `TabControl` (Karteikartenreiter),
- `TreeView-Control` (Baumanzeige) und
- `MultiColumnListbox` (mehrsaltige Listbox).

Aufgrund ihres frühen Entwicklungsstadiums kam es unter einigen Systemen zu Darstellungsproblemen. Dabei wurden Teile der Komponenten nicht korrekt angezeigt (z.B. Scroll-

---

<sup>19</sup> verfügbar unter <http://www.cdt.luth.se/peppar/java/InetAddress/>

balken) bzw. deren Inhalt nicht aktualisiert. Mit Hilfe der Entwicklerforen auf dem Internet-News-Server der Firma Symantec konnten einige der dringendsten Probleme durch Patches gelöst werden. Andere Fehler ließen sich durch Workarounds und Änderungen am mitgelieferten Quellcode der Komponenten beheben.

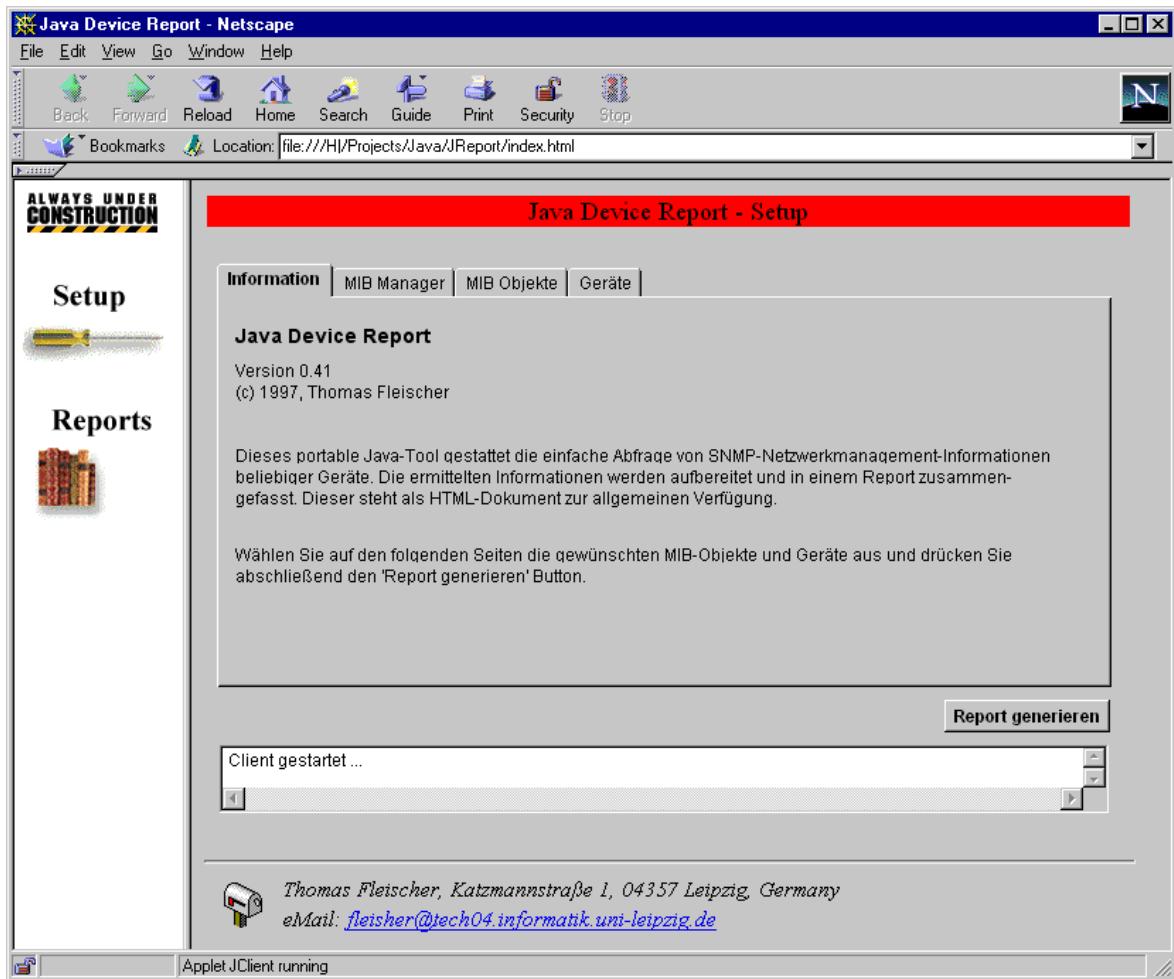
### **5.2.3 Applikationsbeschreibung**

Die Voraussetzung für die Benutzung der Applikation JReport ist natürlich die korrekte Installation und Konfiguration der Anwendung. Dazu gibt eine Kurzbeschreibung auf der beiliegenden Diskette in der Anlage zur Arbeit entsprechende Hinweise.

Die Anwendersitzung beginnt mit dem Start des Applikations-Servers mittels der Java-Laufzeitumgebung und dem Öffnen des Applikations-Clients im Java-fähigen Web-Browser. Der Benutzer wählt für die Dokumentation von Netzkomponenten die gewünschten Managementobjekte aus und gibt die IP-Adressen der abzufragenden Geräte an. Aus diesen Informationen generiert der Applikations-Server entsprechende SNMP-Anfragen und erstellt aus den Antworten der Geräte einen Bericht. Dieser wird anschließend in einem Fenster des Web-Browsers dargestellt.

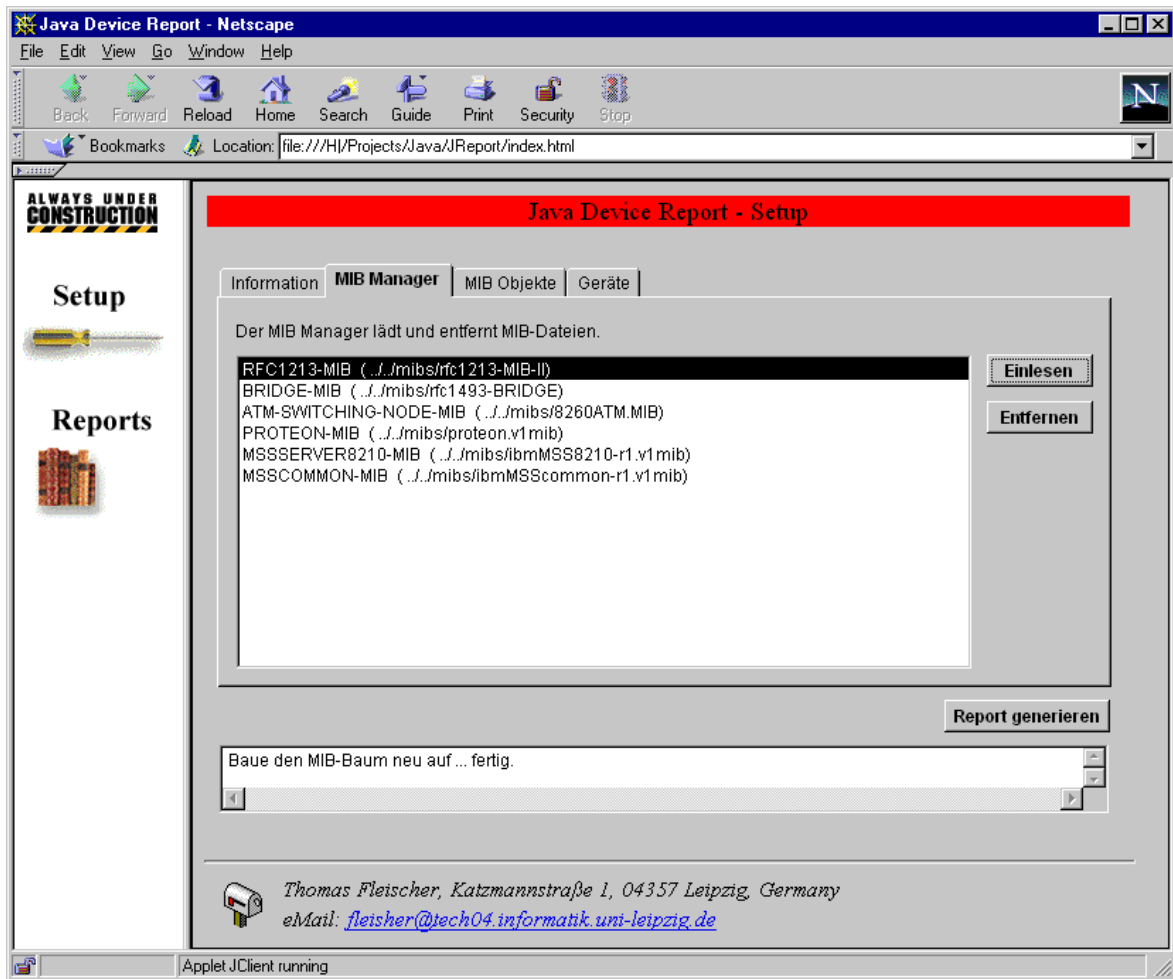
Die Abbildungen auf den folgenden Seiten zeigen eine solche Sitzung aus der Sicht des Benutzers. Dabei werden die einzelnen Konfigurationsschritte in den unterschiedlichen Dialogmasken erläutert. Für die Demonstration des Applikations-Clients kommt hier der Netscape Navigator in der Version 4 zum Einsatz. Die Darstellung unter anderen Browsern und Betriebssystemen kann eventuell geringfügig abweichen.

Die Ansicht der Benutzerschnittstelle unterteilt sich grob in zwei Bereiche: Im linken Bildteil befindet sich ein Navigator, der Verknüpfungen zur generierten Berichtsübersicht („Reports“, siehe Abbildung 28) und zu den Konfigurationsdialogen der Anwendung („Setup“) enthält. Der rechte Bildteil enthält die Masken für die einzelnen Konfigurationsschritte, die über einen Karteikartenreiter (*TabControl*) ausgewählt werden. Zusätzlich befindet sich im unteren rechten Bildbereich ein Statusfenster, in dem Meldungen über den aktuellen Zustand erscheinen.



**Abbildung 24: Die Anwendung JReport - Startbildschirm**

Die Abbildung 24 zeigt den Startbildschirm von JReport, der neben dem Namen und der Versionsnummer der Anwendung einige Informationen für den Gebrauch enthält. Er hat nur informativen Charakter und wird im weiteren Verlauf nicht benötigt.



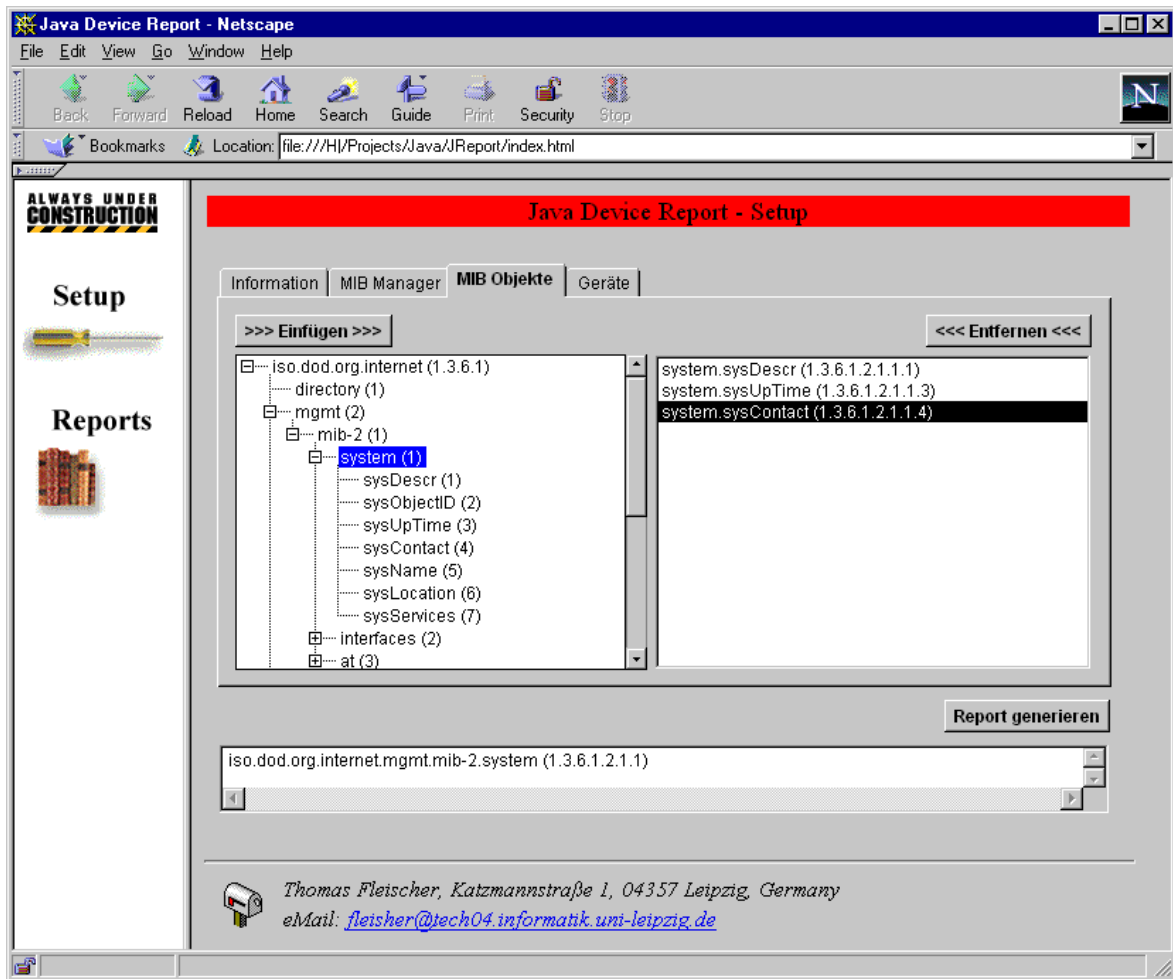
**Abbildung 25: Die Anwendung JReport - Auswahl der MIB-Dateien**

Durch die Anwahl des Karteikartenreiters „MIB-Manager“ erreicht man die erste Konfigurationsmaske. Hier werden die MIB-Dateien verwaltet, d.h. MIBs in die Anwendung eingelesen bzw. bereits geparte MIBs aus der Anwendung entfernt. Die MIB-Objekte der eingelesenen MIBs können in der nächsten Maske ausgewählt werden.

Die Liste in der Abbildung 25 zeigt die bereits vorhandenen MIBs. Mit der Schaltfläche „Entfernen“ kann der aktuell ausgewählte Eintrag aus der Liste gelöscht und somit aus der Anwendung entfernt werden. Die Schaltfläche „Einlesen“ öffnet eine neue Liste, in der alle verfügbaren MIB-Dateien aus dem Verzeichnis<sup>20</sup> <Installationsverzeichnis>\mibs angezeigt werden. Die ausgewählte MIB-Datei wird eingelesen und zur Liste der erfolgreich geparten MIBs hinzugefügt.

Nach jeder Veränderung im „MIB-Manager“ werden die Informationen über die vorhandenen und zur Auswahl stehenden MIB-Objekte aktualisiert.

<sup>20</sup> Neue MIB-Dateien, z.B. herstellerspezifische MIBs, müssen vom Anwender in dieses Verzeichnis kopiert werden. Sie stehen anschließend dem Benutzer sofort zur Verfügung.

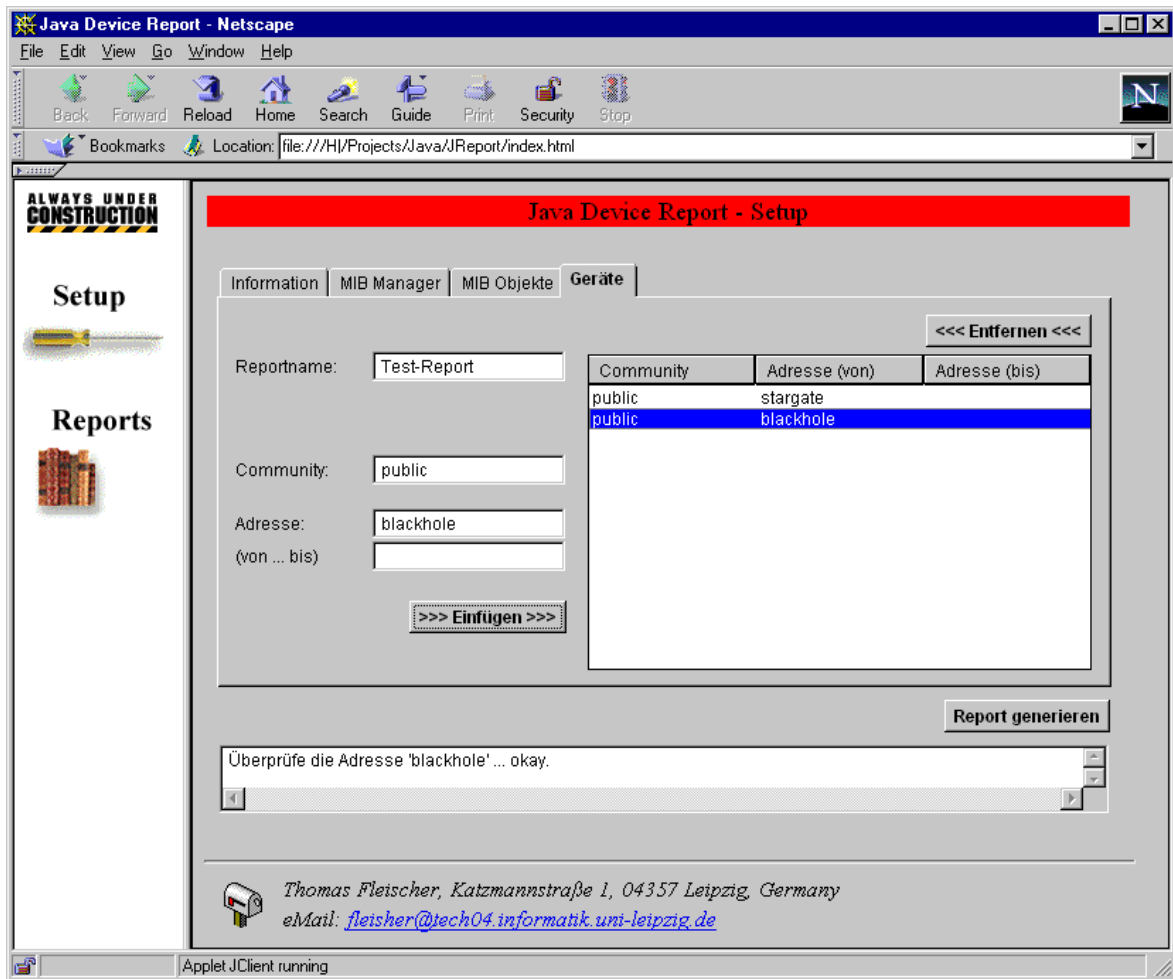


**Abbildung 26: Die Anwendung JReport - Auswahl der MIB-Objekte**

Die Abbildung 26 zeigt die Maske „MIB-Objekte“. Hier werden die gewünschten Elemente für den zu generierenden Report ausgewählt, d.h. die selektierten MIB-Objekte werden anschließend von den Geräten abgefragt und die zurückgelieferten SNMP-Informationen in den Bericht übernommen.

Dazu stellt der linke Bereich des Konfigurationsdialogs alle vorhandenen MIB-Objekte mittels eines TreeView-Controls in einem Baum dar. Die Struktur und Anordnung der Elemente entspricht der Hierarchie der Objekte in den MIBs. Da die Darstellungsfläche begrenzt ist, muß sich die Elementbezeichnung auf den eigentlichen Objektnamen (z.B. sysDescr) beschränken. Der vollständige Name des ausgewählten MIB-Objektes (z.B. iso.org.dod.internet.mgmt.-mib-2.system.sysDescr) wird im Statusfenster angezeigt.

Über die Schaltfläche „Einfügen“ wird das aktuelle Bauelement in die Auswahlliste (rechts) übernommen. Die Schaltfläche „Entfernen“ löscht entsprechend das aktuelle Element der Objektliste.



**Abbildung 27: Die Anwendung JReport - Konfiguration der Geräteadressen**

Über den Karteikartenreiter „Geräte“ erreicht man den letzten Konfigurationsdialog der Anwendung (siehe Abbildung 27). Hier werden die Geräteadressen und die zugehörigen SNMP-Community-Namen der abzufragenden Netzwerkkomponenten festgelegt. Um die Abfrage von Gerätegruppen zu vereinfachen, können auch Adreßbereiche (von - bis) eingegeben werden. Die Verwaltung der Geräteliste erfolgt über die bereits bekannten Schaltflächen „Einfügen“ und „Entfernen“. Zusätzlich ist über das Eingabefeld „Reportname“ die Vergabe einer aussagekräftigen Bezeichnung des zukünftigen Berichts möglich.

Nach Abschluß aller Konfigurationsschritte wird über die Schaltfläche „Report generieren“ (rechts über dem Statusfenster) die eigentliche Berichtserzeugung gestartet. Alle vom Anwender eingegebenen Konfigurationsdaten werden an den Applikations-Server übertragen und dort in entsprechende SNMP-Anfragen für die gewünschten Netzkomponenten konvertiert. Die Antworten der Geräte werden erfaßt, aufbereitet und in einem Report im Dokumentenformat HTML zusammengeführt. Der generierte Dateiname wird an den Client übertragen und der Bericht in einem zweiten Fenster des Web-Browsers dargestellt (siehe Abbildung 29).



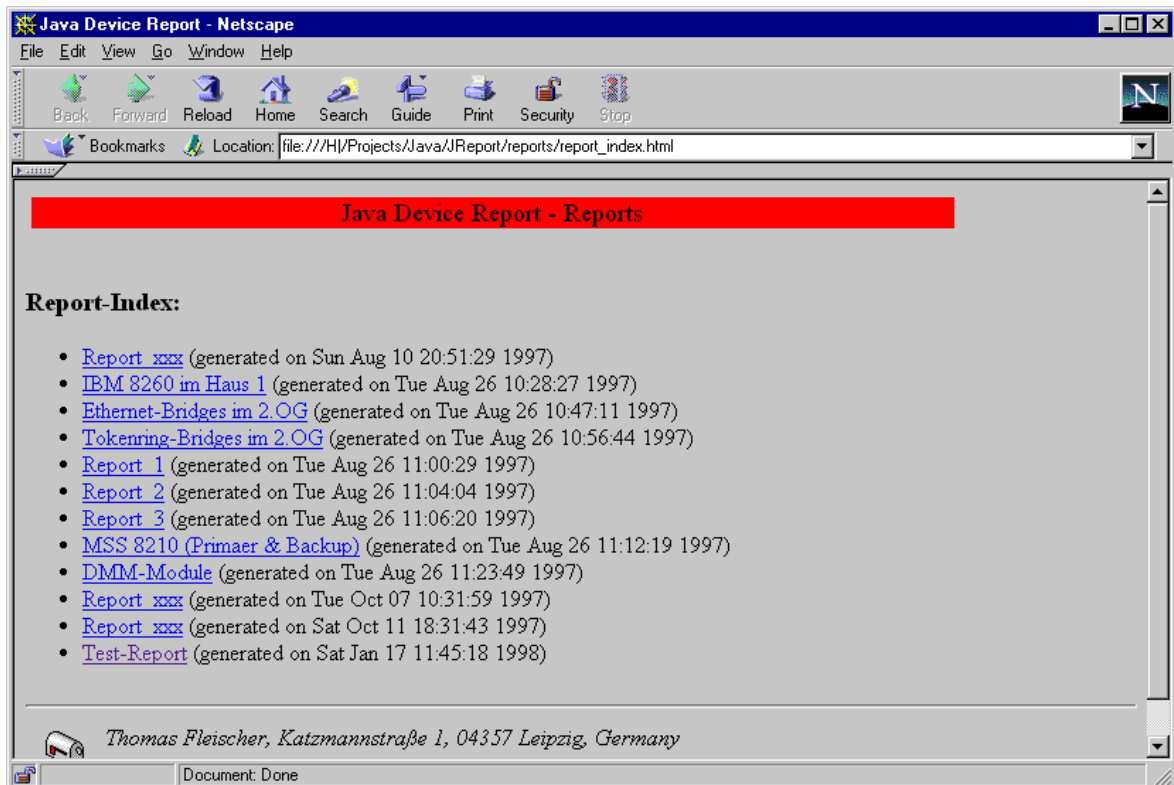


Abbildung 28: Die Anwendung JReport - generierte Berichtsübersicht

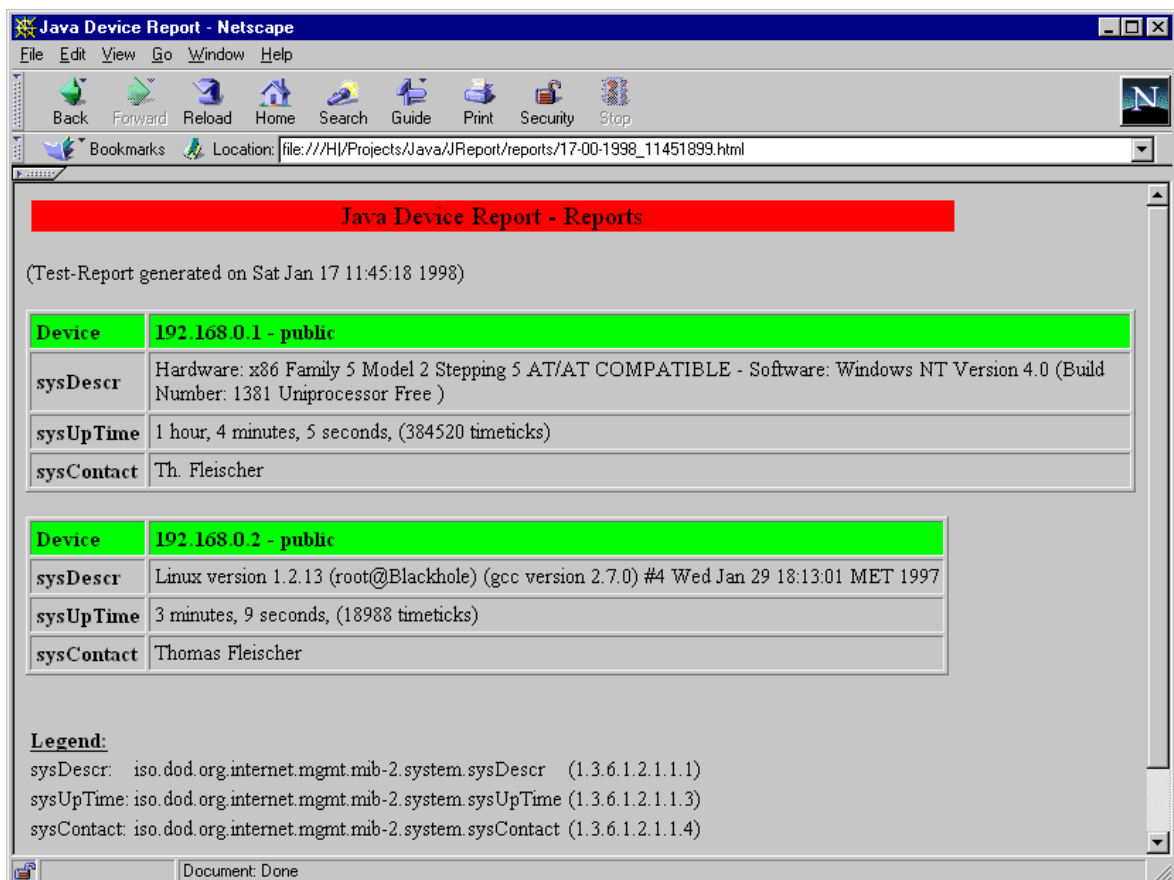


Abbildung 29: Die Anwendung JReport - Beispiel eines erstellten Berichts

Die Abbildung 28 zeigt die automatisch generierte Berichtsübersicht, die über den Navigator auf der linken Seite des Applikationsfensters erreicht werden kann. Die Übersicht enthält alle bisher generierten Berichte in chronologischer Ordnung. Auf einzelne Reports kann über erstellte Links zugegriffen werden. Ein Beispiel eines solchen Reports wird in der Abbildung 29 dargestellt.

#### 5.2.4 Getestete Umgebungen

Die Anwendung JReport konnte sowohl auf dem lokalen Entwicklungssystem als auch auf Rechnern des Instituts für Informatik an der Universität Leipzig und im Umfeld der Norddeutschen Landesbank erfolgreich getestet werden. Dabei wurde die Anwendung unter den in Tabelle 13 aufgelisteten Systemen ausgeführt.

Betriebssystem	Web-Browser
Microsoft Windows NT 4.0	Netscape Navigator 3.0 und Communicator 4.0
Microsoft Windows NT 4.0	Microsoft Internet Explorer 3.0
Microsoft Windows 95	Microsoft Internet Explorer 3.0
IBM AIX 4.1	Netscape Navigator 3.0
IBM OS/2 Warp 4	Netscape Navigator 2.02
Linux, Kernelversion 2.0	Netscape Navigator 3.0

**Tabelle 13: Getestete Umgebungen der Applikation JReport**

Auf keinem der getesteten Systeme waren Änderungen des Java-Programmcodes notwendig. Es wurden lediglich Anpassungen an die jeweils vorhandene Java-Laufzeitumgebung vorgenommen. Diese Einstellungen bezogen sich auf das Setzen von Umgebungsvariablen und die Korrektur eines Shell-Scriptes.

Es ist somit zu erwarten, daß die Applikation auch auf bisher nicht getesteten Umgebungen eingesetzt werden kann. Speziell bei anderen UNIX-Derivaten, z.B. HP-UX, SUN OS, IRIX u.a. sollten keine unvorhergesehenen Probleme auftauchen.

## 6 Zusammenfassung und Ausblick

Die Aufrechterhaltung der Funktionstüchtigkeit von Netzwerken und die Sicherung der Verfügbarkeit vorhandener Netzressourcen erfordern den Einsatz eines umfassenden Netzwerkmanagements. Doch trotz hochspezialisierter Softwareprodukte ergeben sich in diesem Bereich immer wieder Aufgabenfelder, die von den existierenden Lösungen nicht oder nur mangelhaft abgedeckt werden. Neue Entwicklungen, wie Java, geben zudem die Chance, ein plattformunabhängiges Management zu realisieren, das flexibel auf unterschiedlichen Architekturen und Systemen eingesetzt werden kann.

Die Arbeit stellt mit der Applikation JReport das Beispiel einer Anwendung vor, die sich für die Dokumentation von Netzkomponenten eignet. Auf Basis des Netzwerkmanagementprotokolls SNMPv1 wird ein plattformunabhängiger Reportgenerator entworfen und realisiert, der Informationen über konfigurierbare Gerätekategorien abfragen kann. Seine Implementierung in Java demonstriert die Fähigkeiten dieses Systems, das aus einer Kombination von Programmiersprache und Laufzeitumgebung besteht und zeigt dessen Einsatzmöglichkeiten für reale Anwendungen.

Die Schwierigkeiten während der Entwicklung der Anwendung JReport sind häufig auf die verwendete Java-Version 1.0.2 und den frühen Entwicklungsstand zusätzlicher Komponenten zurückzuführen. Die neue Java-Version 1.1.x beseitigt die bekannten Problemstellen der alten Version und bietet zudem zahlreiche Erweiterungen und Neuerungen in der Java-API, z.B. JDBC, JavaBeans und JAR-Dateien. Eine dieser neuen Komponenten ist in Bezug auf die Anwendung JReport besonders interessant. Mit RMI (*Remote Method Invocation*) bietet Java 1.1 eine Schnittstelle, mit der Methoden einer Klasse auf einem anderen Rechner aufgerufen werden können. Damit können verteilte Anwendungen sicher und einfach konstruiert werden. Die Applikation benötigt lediglich eine Referenz auf das entfernte Objekt, den Rest (z.B. Objektserialisierung und Datentransfer) erledigt der RMI-Server.

Mit der RMI-Schnittstelle könnte die bisherige Kommunikation zwischen Applikations-Client und Applikations-Server erheblich vereinfacht werden. Die aufwendige RPC-Implementierung wäre beim Einsatz von RMI überflüssig. Eventuell könnte auch die derzeitige Aufgabenteilung zwischen Client und Server gänzlich entfallen. Konfigurierbare Sicherheitsrestriktionen für Java-Applets im Web-Browser könnten dem Applikations-Client wesentlich mehr Funktionalität ermöglichen. Java 1.1 bietet mit dem neuen Archivformat JAR (*Java Archive*) die Möglichkeit, Applets digital zu signieren, so daß geprüften und vertrauenswürdigen Anwendungen mehr Freiheiten eingeräumt werden. Unter diesen Bedingungen könnten Aufgaben, die derzeit noch vom Applikations-Server ausgeführt werden müssen, auf den Client übertragen werden. Letztendlich wäre die gesamte Anwendung JReport als Java-Applet realisierbar; ein Schritt, der die weitere Entwicklung erheblich vereinfachen würde.

Viele der neuen Konzepte und Ideen scheitern derzeit noch an der mangelhaften Unterstützung von Java 1.1 in den Web-Browsern. Diese implementieren bisher nur Teile der neuen Java-Version oder hängen der aktuellen Java-Entwicklung bei Sun erheblich hinterher. Eine kurzfristige Abhilfe bietet Sun mit dem Plugin „Java Activator“ für Web-Browser. Es enthält eine JVM mit vollständiger Java 1.1-Unterstützung und ist derzeit für den Microsoft Internet Explorer und den Netscape Navigator unter Win32, Solaris und Linux verfügbar. Leider hat das Plugin auch einen entscheidenden Nachteil: Es ersetzt die JVM des Browsers nicht, sondern läuft parallel zu ihr. Um das Plugin benutzen zu können, müssen die Java-Applets im HTML-Dokument mit neuen HTML-Tags (`<OBJECT>` oder `<EMBED>`) gekennzeichnet werden. Der Standard-Tag `<APPLET> . . . </APPLET>` ruft weiterhin die JVM des Web-Browsers auf. Eine Lösung des Java 1.1-Problems kündigte indes Netscape an. Man wolle in zukünftigen Versionen des eigenen Web-Browsers die JVM entfernen und durch eine Plugin-Schnittstelle namens „OpenJava API“ ersetzen. Hiermit könne der Anwender selbst die aktuellste und schnellste JVM von Sun oder einem anderen Anbieter benutzen. Damit würde der Entwicklungsaufwand bei Netscape erheblich reduziert und bisherige Verzögerungen bei der Freigabe einer neuen JVM entfallen.

Aber auch ohne eine mögliche Umstellung auf Java 1.1 verbleiben noch zahlreiche Möglichkeiten für Verbesserungen und Erweiterungen zukünftiger Versionen der Anwendung JReport. So könnte die SNMP-Library demnächst die weiterentwickelten SNMP-Versionen unterstützen, z.B. SNMPv3, das als neuer Standard des Netzwerkmanagements etabliert werden soll. Ebenfalls sinnvoll wäre eine Integration der MIB-Library in die SNMP-Lib. Bei einer Vereinigung der bisher getrennten Schnittstellen könnten Arbeitsschritte, die derzeit noch vom Anwendungsentwickler durchgeführt werden müssen, entfallen. Die Kombination beider Komponenten könnte Parameter flexibler verarbeiten (z.B. OIDs in Form von MIB-Objektnamen) und Ausgaben von SNMP-Informationen automatisch in eine verständliche Darstellung konvertieren. Auch die Anwendung selbst bietet noch Spielraum für zahlreiche Veränderungen. So könnte die Ausgabeverarbeitung im Applikations-Server, die derzeit die Generierung der Berichte im Dokumentenformat HTML unterstützt, erweitert werden. Eine Schnittstelle für Ausgabefilter würde die Möglichkeit bieten neue Generatorfunktionen flexibel einzubinden und so weitere Ausgabeformate zu unterstützen.

Die Applikation JReport ist ein Beispiel für den Einsatz des Java-Systems im Bereich des Netzwerkmanagements. Es demonstriert die Anwendungsmöglichkeiten, die sich durch die Verwendung des plattformunabhängigen Laufzeitsystems ergeben. So kann die in dieser Arbeit entwickelte Softwarelösung, im Gegensatz zu herkömmlichen nativen Anwendungen, ohne vorherige Änderungen am Quellcode oder Neuübersetzungen auf unterschiedlichen Architekturen und Systemen eingesetzt werden. Mit dieser Eigenschaft empfiehlt sich der Einsatz der Applikation JReport in heterogenen Umgebungen, bzw. Arbeitsbereichen, in denen

das Arbeitsgerät Computer häufigen Änderungen unterliegt. Dies trifft z.B. auf Servicetechniker im Bereich der Wartung zu.

Insgesamt stellt die Arbeit die notwendigen Hilfsmittel für die Entwicklung kleiner spezialisierter Tools auf dem Gebiet des plattformunabhängigen Netzwerkmanagements zur Verfügung. Die Verwendungsmöglichkeiten dieser Komponenten wurden am Beispiel der Anwendung JReport demonstriert.

## **Abkürzungsverzeichnis**

ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation 1
AWT	Abstract Window Toolkit
BER	Basic Encoding Rules
CCITT	Comité Consultatif International de Téléphonie et Télégraphie
CERN	Centre Européen de Recherches Nucléaires
CLNS	OSI Connectionless-Mode Network Service
CMIP	OSI Common Management Information Protocol
CMOT	CMIP over TCP
CONS	OSI Connection-Oriented Network Service
CPU	Central Processing Unit
DES	Data Encryption Standard
DIS	Draft International Standard
EGP	Exterior Gateway Protocol
FTP	File Transfer Protocol
GUI	Graphical User Interface
HEMS	High-Level Entity Management System
HMP	Host Monitoring Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPX	Novell Internetwork Packet Exchange
IS	International Standard

ISO	International Organization for Standardization
JAR	Java Archive
JDK	Java Development Kit
JIT	Just-In-Time-Compiler
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LAN	Local Area Network
LSB	Least Significant Bit
MAN	Metropolitan Area Network
MD5	Message Digest Algorithm
MIB	Management Information Base
MIT	Massachusetts Institute of Technology
MSB	Most Significant Bit
NCSA	National Center for Supercomputing Applications
NMS	Network Management System
OID	Object Identifier
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
RFC	Request for Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAP	Service Access Point
SGMP	Simple Gateway Monitoring Protocol
SMFA	Systems Management Functional Area
SMI	Structure of Management Information
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SNMPv1	SNMP Version 1
SNMPv2	SNMP Version 2

SNMPv2p	party-based SNMPv2
SNMPv2c	community-based SNMPv2
SNMPv2u	user-based SNMPv2
SNMPv3	SNMP Version 3
TCP	Transmission Control Protocol
TLV	Tag, Length, Value
TMN	Telecommunication Management Network
UDP	User Datagram Protocol
VM	Virtual Machine
WWW	World Wide Web



## Literatur- und Quellenverzeichnis

(Alle aufgeführten Internet-Quellen basieren auf dem Stand von November 1997.)

- [CAM96] Campoine, Mary; Walrath, Kathy: „The Java Tutorial - Object-Oriented Programming for the Internet“. Addison-Wesley 1996.
- [CMU12] Carnegie-Mellon University, 1992.  
<ftp://ftp.net.cmu.edu/pub/cmu-snmp1.2u.tar.Z>.
- [DAL97] Dalheimer, Matthias Kalle: „Java Virtual Machine - Sprache, Konzept, Architektur“. O'Reilly Verlag 1997.
- [ECK97] Eckel, Bruce: „Thinking in Java“. Revision 4, Februar 1997.  
<http://www.eckelobjects.com/eckel>.
- [FRE96] Freeman, Adam; Ince, Darrel: „Active Java - Object-Oriented Programming for the World Wide Web“. Addison-Wesley 1996.  
<http://www.aw.com/cseng/authors/freeman.a/actjava/actjava.sup.html>.
- [HEI95] Heitlinger, Paulo: „Netzwerk-Management - Komplettlösungen und Tools“. Thomson Publishing 1995.
- [JAN93] Janssen, Rainer; Schott Wolfgang: „SNMP - Konzepte, Plattformen, Verfahren“. DATACOM-Verlag 1993.
- [LIN96] Linnhoff-Popien, Claudia; Reichl, Peter: „Netzwerkmanagement - Skript zu den Vorlesungen an der RWTH Aachen und der Universität GH Essen“. Verlag der Augustinus Buchhandlung 1996.
- [PER93] Perkins, T. David: „Understanding SNMP MIBs“. Revision 1.1.17, September 1993. <ftp://ftp.net.cmu.edu/pub/UnderstandingSNMPv1Mibs.ps>.
- [RFC1155] Rose, T. Marshall; McCloghrie, K.: „Structure and Identification of Management Information for TCP/IP-based Internets“. RFC 1155, 1990.  
<http://www.cis.ohio-state.edu/htbin/rfc/rfc1155.html>.
- [RFC1157] Case, J.; Fedor, M.; Schoffstall, M.; Davin, J.: „A Simple Network Management Protocol (SNMP)“. RFC 1157, 1990.  
<http://www.cis.ohio-state.edu/htbin/rfc/rfc1157.html>.
- [RFC1212] Rose, T. Marshall; McCloghrie, K.: „Concise MIB Definitions“. RFC 1212, 1991.  
<http://www.cis.ohio-state.edu/htbin/rfc/rfc1212.html>.
- [RFC1213] McCloghrie, K.; Rose, T. Marshall: „Management Information Base for Network Management of TCP/IP-based Internets: MIB-II“. RFC 1213, 1991.  
<http://www.cis.ohio-state.edu/htbin/rfc/rfc1213.html>.

- [ROS93] Rose, T. Marshall: „Einführung in die Verwaltung von TCP/IP-Netzen“. Carl Hanser Verlag und Prentice Hall 1993.
- [ROS94] Rose, T. Marshall: „The Simple Book - An Introduction to Internet Management“. Prentice Hall 1994, 2. Auflage.
- [SEL95] Sellin, Rüdiger: „TMN - Die Basis für das Telekom-Management der Zukunft“. R. v. Decker's Verlag 1995.
- [SLO94] Sloman, Morris: „Network and Distributed Systems Management“. Addison-Wesley 1994.
- [STA96] Stallings, William: „SNMP, SNMPv2 and RMON - Practical Network Management“. Addison-Wesley 1996, 2. Auflage.
- [STE96] Stevens, W. Richard: „TCP/IP Illustrated, Volume 1“. Addison-Wesley 1996, 7. Auflage.
- [SUN95] Sun Microsystems: „The Java Language: An Overview“, 1995.  
<http://java.sun.com/docs/overviews/java/java-overview-1.html>.
- [WEB96] Weber, Joseph; et al.: „Special Edition Using Java, 2nd Edition“. QUE 1996, 2. Auflage.

## Anhang A: Verwendete Hard- und Software

Folgende Hard- und Software wurden während der Entwicklung von „JReport“ benutzt.

### Hardware:

<b>Entwicklungsplattform</b>	PC Pentium 100 MHz, 64 MB RAM, Ethernet-Netzwerkschnittstelle
<b>Testsystem 1</b>	PC 486DX 33 MHz, 8 MB RAM, Ethernet-Netzwerkschnittstelle
<b>Testsystem 2</b>	PowerPC 100 MHz, 128 MB RAM, Ethernet-Netzwerkschnittstelle und 100 MBit ATM-Interface

### Betriebssysteme:

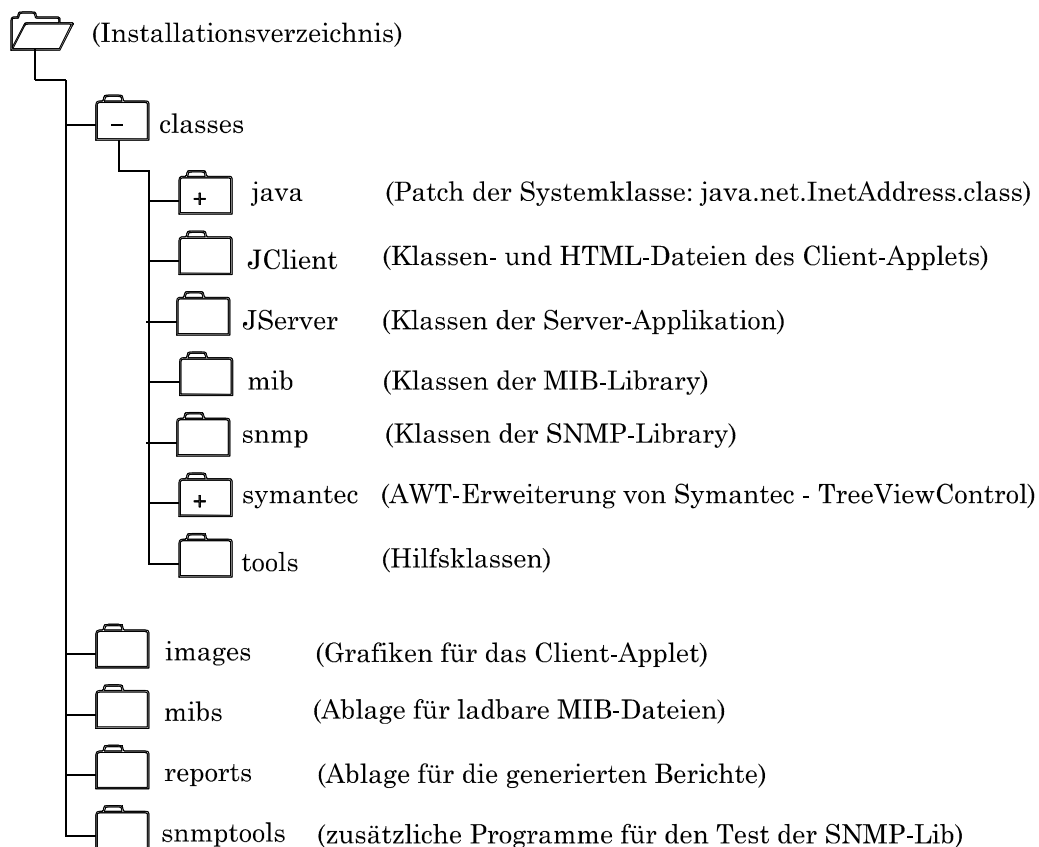
<b>Entwicklungsplattform</b>	MS Windows NT 4.0 Workstation
<b>Testsystem 1</b>	Linux, Kernelversion 1.2.13
<b>Testsystem 2</b>	IBM AIX 4.1.4

### verwendete Softwarepakete:

- SUN / JavaSoft Java Development Kit JDK 1.02
- Symantec Visual Café 1.0
- Webcetera Kawa-IDE 2.5
  
- Netscape Navigator 3
- Netscape Communicator 4
- MS Internet Explorer 3
  
- CMU SNMP Package „cmu-snmp1.2u.tar.Z“
- CMU SNMP Package „cmu-snmp-linux-3.3-src.tar.gz“
- SNMP-Dienst für MS Windows NT
- IBM NetView/6000

## Anhang B: Quelltexte

Aufgrund des nicht unerheblichen Umfangs der vorhandenen Quelltexte für die Applikation „JReport“ (ca. 300 KB) muß auf einen Abdruck an dieser Stelle verzichtet werden. Die entsprechenden Dateien befinden sich zusammen mit einer ablauffähigen Version der Anwendung sowie einer kurzen Installationsbeschreibung auf einer Diskette in der Anlage zu dieser Arbeit. Dort sind die Quelltexte in dem ZIP-Archiv „jreport.zip“ abgelegt, das mit einem entsprechenden Programm, z.B. Unzip oder WinZIP, dekomprimiert werden kann. Nach diesem Schritt sollte folgende Verzeichnisstruktur vorzufinden sein:



Sämtliche Quelltexte der Applikation „JReport“ befinden sich zusammen mit den übersetzten Klassendateien im Verzeichnis „classes“.

## **Danksagung**

Hiermit möchte ich allen Personen danken, die mir dabei geholfen haben, die vorliegende Arbeit zu schreiben.

Besonders danke ich Herrn Dipl.-Ing. J. Hoffmann. Durch sein Engagement erhielt ich die Möglichkeit, diese Arbeit bei der IBM Deutschland Informationssysteme GmbH zu beginnen. Seine ständige Unterstützung in Form ausführlicher Diskussionen trug zur Schärfung meines Verständnisses für die Problematik „Netzwerkmanagement“ bei und gab wertvolle Anregungen für den Entwurf der Anwendung „JReport“. Nach seinem Ausscheiden aus dem Unternehmen übernahm Herr M. Frese unbürokratisch die Funktion des Betreuers und Ansprechpartners bei der IBM, wofür ich mich noch einmal bedanken möchte.

Weiterhin gilt mein Dank den Mitarbeitern der Abteilung Technische Informatik des Instituts für Informatik der Universität Leipzig. Wertvolle Tips und Anregungen erhielt ich von Herrn Professor Dr.-Ing. W. G. Spruth sowie Herrn Dr. K. Hänßgen, der diese Arbeit tatkräftig unterstützte.

Verschiedene Personen haben das Manuskript dieser Arbeit gelesen und zahlreiche Korrektur- und Verbesserungsvorschläge gemacht. Hierfür möchte ich besonders Herrn M. Müller danken.

Vor allem danke ich aber meinen Eltern, deren fortwährende Unterstützung diese Arbeit erst möglich gemacht hat.

## **Erklärung**

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, \_\_\_\_\_

\_\_\_\_\_  
Thomas Fleischer